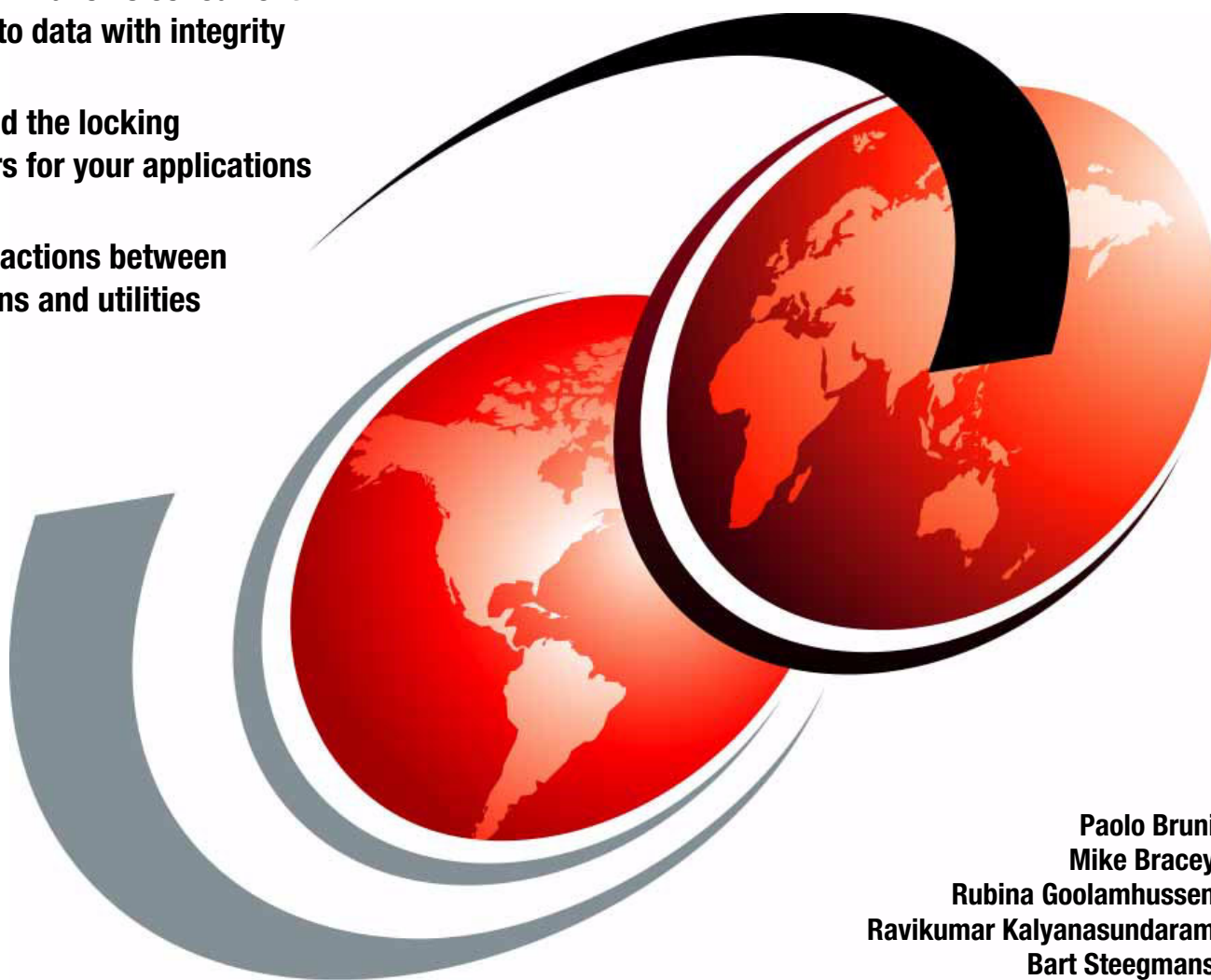


# DB2 9 for z/OS: Resource Serialization and Concurrency Control

See how DB2 allows concurrent  
accesses to data with integrity

Understand the locking  
parameters for your applications

Tune interactions between  
applications and utilities



Paolo Bruni  
Mike Bracey  
Rubina Goolamhussen  
Ravikumar Kalyanasundaram  
Bart Steegmans

**Redbooks**





International Technical Support Organization

**DB2 9 for z/OS: Resource Serialization and  
Concurrency Control**

December 2009

**Note:** Before using this information and the product it supports, read the information in “Notices” on page xix.

**First Edition (December 2009)**

This edition applies to Version 9.1 of IBM DB2 for z/OS (program number 5635-DB2).

**© Copyright International Business Machines Corporation 2009. All rights reserved.**

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Figures</b> .....	xi
<b>Tables</b> .....	xiii
<b>Examples</b> .....	xv
<b>Notices</b> .....	xix
Trademarks .....	xx
<b>Preface</b> .....	xxi
The team who wrote this book .....	xxi
Become a published author .....	xxiii
Comments welcome .....	xxiii
<b>Part 1. Concurrency and integrity</b> .....	1
<b>Chapter 1. Serialization</b> .....	3
1.1 Introduction .....	4
1.2 Transaction management .....	5
1.3 The reasons for serialization .....	5
1.4 Isolation levels .....	7
1.5 Unit of work and unit of recovery .....	9
1.5.1 Online transaction processing .....	12
1.5.2 Batch and distributed environments .....	12
1.6 Serialization mechanisms used by DB2 .....	12
<b>Chapter 2. Transaction locking</b> .....	13
2.1 Transaction locking .....	14
2.1.1 Lock size .....	14
2.1.2 Lock state or lock mode .....	16
2.1.3 Lock duration .....	19
2.1.4 Single and two-phase commit write considerations .....	23
2.1.5 Dynamic statement concurrency .....	23
2.1.6 LOB locks .....	25
2.1.7 XML locks .....	26
2.1.8 Global temporary tables .....	26
2.1.9 Global transactions .....	26
2.2 Locking control options .....	27
2.2.1 Skip locked data .....	27
2.2.2 Lock promotion .....	30
2.2.3 Lock escalation .....	30
2.2.4 Lock suspension and timeout .....	31
2.2.5 Deadlock .....	32
2.3 Lock avoidance .....	33
2.3.1 Lock avoidance control .....	36
<b>Chapter 3. Serialization techniques</b> .....	41
3.1 Index specific lock .....	42
3.2 Mass delete lock .....	44
3.3 The Internal Resource Lock Manager (IRLM) .....	44

3.3.1 IRLM principles of operation . . . . .	45
3.3.2 Types of lock request . . . . .	45
3.3.3 Lock processing . . . . .	47
3.3.4 IRLM service class definition in Workload Manager (WLM) . . . . .	49
3.3.5 DB2 and Workload Manager application process prioritization. . . . .	51
3.4 DB2 subsystem object locking . . . . .	51
3.4.1 Locks on the DB2 catalog and directory . . . . .	52
3.4.2 Locks on skeleton tables. . . . .	52
3.4.3 Locks on database descriptors . . . . .	52
3.5 Claims and drains . . . . .	53
3.5.1 Claims. . . . .	54
3.5.2 Drains . . . . .	54
3.5.3 Interaction between claims and drains . . . . .	55
3.6 Partition independence . . . . .	56
3.7 Restricted states . . . . .	56
3.8 Latches . . . . .	56
3.8.1 Buffer pool data and index page latching . . . . .	56
3.8.2 DB2 latches. . . . .	58

## **Part 2. Application concurrency and lock optimization . . . . . 63**

<b>Chapter 4. Database design considerations. . . . .</b>	<b>65</b>
4.1 Table space considerations . . . . .	66
4.1.1 Type of table space. . . . .	66
4.1.2 Page size . . . . .	67
4.1.3 LOCKSIZE . . . . .	67
4.1.4 LOCKMAX . . . . .	69
4.1.5 FREEPAGE and PCTFREE . . . . .	71
4.1.6 MAXROWS. . . . .	71
4.1.7 COMPRESS YES. . . . .	72
4.1.8 DEFINE YES. . . . .	72
4.2 Table design considerations . . . . .	72
4.2.1 FOREIGN KEY references-clause . . . . .	72
4.2.2 VOLATILE table . . . . .	73
4.2.3 MQT instead of denormalization . . . . .	73
4.2.4 NOT LOGGED facility . . . . .	73
4.2.5 APPEND YES. . . . .	73
4.2.6 Global temporary tables . . . . .	74
4.3 Index design and locking considerations . . . . .	75
4.3.1 FREEPAGE and PCTFREE . . . . .	75
4.3.2 Page size . . . . .	75
4.3.3 COMPRESS YES. . . . .	75
4.3.4 Data partitioned secondary indexes . . . . .	76
4.3.5 CLUSTER . . . . .	76
4.3.6 RANDOM index . . . . .	76
4.3.7 Instead of trigger . . . . .	76
4.4 Sequence object . . . . .	77
4.4.1 Sequence objects . . . . .	78
4.5 Database consideration . . . . .	80
4.6 Hot spot scenarios . . . . .	80
4.6.1 Hot data page problem . . . . .	81
4.6.2 Hot row problem . . . . .	82
4.6.3 Sequential numbers in DB2 tables . . . . .	84

4.7 Database design recommendations for concurrency . . . . .	84
<b>Chapter 5. Application design . . . . .</b>	<b>87</b>
5.1 SQL design . . . . .	88
5.1.1 Types of SQL processing and locks requested by DB2 . . . . .	88
5.1.2 Type of table spaces and locks acquired by DB2 . . . . .	99
5.2 Optimizing concurrency and locking . . . . .	100
5.2.1 When will the lock be acquired . . . . .	100
5.2.2 Will a lock be required. . . . .	101
5.2.3 How restrictive will the lock be . . . . .	101
5.2.4 When will the lock be released . . . . .	101
5.3 Locking and concurrency techniques with isolation levels . . . . .	102
5.3.1 Isolation levels . . . . .	103
5.4 Higher level locks . . . . .	105
5.4.1 LOCK TABLE statement. . . . .	105
5.4.2 Lock demotion. . . . .	107
5.5 Read-only queries . . . . .	107
5.5.1 Read with no lock . . . . .	108
5.5.2 Mass delete locks . . . . .	112
5.6 Optimistic concurrency control . . . . .	113
5.6.1 Optimistic locking using ROW CHANGE TIMESTAMP on the table. . . . .	114
5.6.2 Optimistic locking in updateable static scrollable cursors . . . . .	117
5.7 Using the SKIP LOCKED DATA option. . . . .	118
5.8 Unit of recovery and unit of work. . . . .	120
5.8.1 Savepoint . . . . .	121
5.8.2 Commit . . . . .	121
5.8.3 Online processes . . . . .	124
5.8.4 Batch processes and commit frequency . . . . .	125
5.9 Locks acquired by unconventional SQL statements . . . . .	126
5.9.1 MQT REFRESH TABLE statement. . . . .	127
5.9.2 EXCHANGE DATA BETWEEN clone table and base table . . . . .	128
5.9.3 TRUNCATE statement . . . . .	130
5.9.4 Static and dynamic scrollable cursors. . . . .	130
5.9.5 MERGE statement . . . . .	131
5.9.6 Data change within a SELECT statement. . . . .	132
5.10 Concurrency scenarios . . . . .	133
5.10.1 Scenario 1: lock interaction between concurrent SQL statements . . . . .	133
5.10.2 Scenario 2: INSERT timeouts . . . . .	134
5.11 LOB and XML locks . . . . .	135
5.11.1 LOB locks . . . . .	135
5.11.2 XML locks . . . . .	138
5.12 Access path influence on locking . . . . .	138
5.12.1 Index-only access and data-only locking . . . . .	139
5.12.2 Temporary result tables and lock avoidance. . . . .	140
5.12.3 Table space scan with repeatable reads. . . . .	141
5.12.4 EXPLAIN statement . . . . .	141
5.13 CICS: locking considerations . . . . .	143
5.13.1 Protected entry threads. . . . .	143
5.13.2 Pseudo-conversation and locking . . . . .	143
5.14 Distributed application considerations. . . . .	143
5.15 Dynamic SQL considerations . . . . .	145
5.15.1 DBD locks . . . . .	145
5.15.2 Bind option RELEASE(DEALLOCATE) . . . . .	145

5.16	Programming for the Instrumentation Facility Interface. . . . .	146
5.16.1	Locking considerations for IFI. . . . .	146
5.17	Data integrity exposure issues . . . . .	147
5.18	Performance considerations . . . . .	147
5.19	How to prevent locking problems . . . . .	150
5.19.1	Suspension, deadlock, timeout, and lock escalation. . . . .	150
5.19.2	Deadlock avoidance . . . . .	152
5.19.3	Application action after a deadlock or timeout . . . . .	155
5.19.4	Recommendations to prevent locking problems . . . . .	155
5.20	List of common lock types: OMEGAMON PE online monitor . . . . .	159
<b>Chapter 6. Utilities, commands, and SQL. . . . .</b>		<b>161</b>
6.1	Claims and drains for concurrency control . . . . .	162
6.1.1	Claim. . . . .	162
6.1.2	Drains . . . . .	163
6.2	Utility concurrency and compatibility . . . . .	164
6.2.1	Utility locks on the catalog and directory. . . . .	164
6.2.2	Compatibility of utilities . . . . .	164
6.3	Serialization between utilities and SQL. . . . .	165
6.3.1	Scenario for concurrency of utilities and SQL. . . . .	166
6.3.2	Deadlock avoidance between utilities and SQL . . . . .	172
6.4	Partition independence . . . . .	172
6.4.1	Scenario with LOAD PART and concurrent SQL updates . . . . .	174
6.4.2	Utility operations with non-partitioned indexes . . . . .	175
6.5	Online utilities and concurrency . . . . .	176
6.5.1	Transaction locks on online LOAD and UNLOAD utilities. . . . .	176
6.5.2	UNLOAD from image copy data sets . . . . .	176
6.5.3	Online utilities that acquire drain lock . . . . .	177
6.5.4	Online utilities without drain . . . . .	178
6.6	DB2 command considerations . . . . .	179
6.6.1	START command . . . . .	179
6.6.2	STOP command . . . . .	180
6.6.3	BIND command. . . . .	180
6.6.4	Conflicting BIND options. . . . .	190
6.7	Locks on DB2 objects . . . . .	190
6.7.1	Interaction between bind, DDL, and DML. . . . .	191
6.7.2	Contention on the DB2 catalog . . . . .	192
6.7.3	Locks on skeleton tables. . . . .	192
6.7.4	Locks on the database descriptors (DBDs). . . . .	193
6.8	Restricted states . . . . .	193
<b>Chapter 7. System considerations . . . . .</b>		<b>201</b>
7.1	Application and system interaction . . . . .	202
7.2	IRLM start procedure parameters . . . . .	202
7.2.1	Estimating the storage needed for locks. . . . .	204
7.3	DB2 system parameters that directly affect locking . . . . .	204
7.3.1	IRLMRWT: RESOURCE TIMEOUT field . . . . .	204
7.3.2	DEADLOK (part 1): DEADLOCK TIME. . . . .	207
7.3.3	DEADLOK (part 2): DEADLOCK CYCLE . . . . .	207
7.3.4	BMPTOUT: IMS BMP TIMEOUT multiplier. . . . .	207
7.3.5	DLITOUT: DL/I BATCH TIMEOUT multiplier . . . . .	208
7.3.6	UTIMOUT: UTILITY TIMEOUT multiplier . . . . .	208
7.3.7	RETLWAIT: RETAINED LOCK TIMEOUT multiplier . . . . .	208



7.3.8	NUMLKTS: maximum LOCKS PER TABLE (SPACE)	208
7.3.9	NUMLKUS: maximum LOCKS PER USER	209
7.3.10	RRULOCK: U LOCK FOR RR/RS isolation	210
7.3.11	XLKUPDLT: X LOCK FOR SEARCHED Update or Delete.	210
7.3.12	EVALUNC: EVALUATE UNCOMMITTED rows	211
7.3.13	SKIPUNCI: SKIP UNCOMMITTED INSERTS	211
7.3.14	IDTHTOIN: IDLE THREAD TIMEOUT field.	212
7.3.15	LRDRTHLD: LONG-RUNNING READER field	212
7.3.16	URCHKTH: UR CHECK FREQ field	212
7.3.17	URLGWTH: UR LOG WRITE CHECK field	213
7.3.18	PCLOSEN: RO SWITCH CHKPTS field	213
7.3.19	PCLOSET: RO SWITCH TIME field	213
7.3.20	CHKFREQ: CHECKPOINT FREQ field	214
7.4	DB2 system parameters that indirectly affect locking	214
7.4.1	SMFACCT: SMF ACCOUNTING field	214
7.4.2	SMFSTAT: SMF STATISTICS field	215
7.4.3	STATIME: STATISTICS TIME field.	215
7.4.4	SYNCVAL: STATISTICS SYNC field	215
<b>Part 3.</b>	<b>Monitoring and problem determination</b>	<b>217</b>
<b>Chapter 8.</b>	<b>Identifying locking and concurrency problems</b>	<b>219</b>
8.1	Different types of locking and concurrency problems	220
8.1.1	Timeouts	220
8.1.2	Deadlocks	220
8.1.3	Lock escalation	220
8.1.4	Long suspension times	221
8.1.5	Excessive number of locks acquired.	221
8.2	Concurrency warning signs	222
8.2.1	User warnings	222
8.2.2	Console messages	222
8.3	Periodic monitoring for concurrency problems	227
8.3.1	Which information to gather	227
8.3.2	Analyzing DB2 statistics data	228
8.3.3	Analyzing DB2 accounting data	236
8.3.4	Periodic monitoring of DB2 system level concurrency related IFCIDs	243
8.3.5	Exception monitoring using OMEGAMON PE	252
<b>Chapter 9.</b>	<b>Analyzing concurrency problems</b>	<b>255</b>
9.1	The analysis toolbox	256
9.1.1	DB2 commands	256
9.1.2	EXPLAIN.	262
9.1.3	Standard DB2 traces.	265
9.1.4	Detailed DB2 traces	265
9.1.5	Using online monitor	280
9.2	Analysis of a simple deadlock scenario	280
9.2.1	Analyzing the joblogs and syslog	281
9.2.2	Analyzing the deadlock record	282
9.2.3	Checking the DB2 accounting data.	284
9.2.4	Zooming in using the record trace.	287
9.2.5	The complete deadlock picture	299
<b>Part 4.</b>	<b>Data sharing</b>	<b>303</b>

<b>Chapter 10. Global locking</b>	305
10.1 Global locking overview	306
10.2 DB2 data sharing system locking components	309
10.3 Introducing the DB2 lock structure	311
10.4 L-locks versus P-locks	314
10.5 How inter-DB2 data coherency or read-write interest works	315
10.5.1 Inter-DB2 read/read interest	317
10.5.2 Inter-DB2 read/write interest	319
10.5.3 Inter-DB2 write/write interest	321
10.6 Data sharing locking optimizations	322
10.6.1 Lock optimization example	323
10.6.2 Explicit hierarchical locking	325
10.7 Page P-locks	329
10.7.1 Page P-locks usage by row level locking	330
10.7.2 Other usage of page P-locks	332
10.7.3 Other types of P-locks	334
10.8 Pseudo-close and the GBP-dependency	334
10.8.1 A single member loses interest in a page set	335
10.8.2 Multiple members losing interest	337
10.8.3 Reading member quits first	338
10.8.4 Pseudo-close considerations	339
10.9 Managing the DB2 lock structure	341
10.9.1 Definition of the lock structure	341
10.9.2 Lock table size	342
10.9.3 Lock table entries	345
10.9.4 Record list table	347
10.9.5 Modify and retained locks	348
10.10 Global contention	353
10.10.1 Global lock manager	353
10.10.2 Real contention	354
10.10.3 False contention	354
10.10.4 XES contention	356
10.10.5 Managing contention	358
10.11 Timeout and deadlock in a data sharing environment	360
10.12 Lock avoidance in data sharing	362
<b>Chapter 11. Monitoring data sharing locking activity</b>	365
11.1 Monitoring using DB2 commands	366
11.2 Monitoring the lock structure size and entries	369
11.2.1 The -DISPLAY GROUP command	370
11.2.2 z/OS command D XCF	370
11.3 DB2 statistics and accounting traces	372
11.3.1 OMEGAMON PE SCOPE(MEMBER/GROUP) reports	373
11.3.2 DB2 statistics data	374
11.3.3 DB2 accounting data	382
11.4 Monitoring data sharing locks using RMF reports	389
11.5 Data sharing global contention scenarios	392
11.5.1 Global lock avoidance	392
11.5.2 LOCKSIZE ROW versus LOCKSIZE PAGE	394
11.5.3 High INSERT data scenario	398
11.6 Data sharing lock contention analysis	403
11.6.1 Analyzing the accounting information	404
11.6.2 Lock suspension report	408

11.6.3 Analyzing an individual global lock suspension . . . . .	411
<b>Chapter 12. Database and application design in data sharing</b> . . . . .	417
12.1 Database design considerations . . . . .	418
12.1.1 Table space design . . . . .	418
12.1.2 Index design . . . . .	422
12.2 Application design guidelines . . . . .	426
12.2.1 Maximize lock avoidance . . . . .	426
12.2.2 Table space lock duration . . . . .	427
12.2.3 P-locks and isolation levels . . . . .	428
12.2.4 Use uncommitted read isolation . . . . .	428
12.2.5 Sequences and identity columns . . . . .	428
12.2.6 Create member affinity . . . . .	430
12.2.7 IMMEDIATE . . . . .	431
12.3 DB2 utilities in a data sharing environment . . . . .	432
12.3.1 Utilities and partitioned objects . . . . .	433
12.3.2 Utilities and pageset P-locks . . . . .	434
<b>Part 5. Appendixes</b> . . . . .	439
<b>Appendix A. System topology and workload</b> . . . . .	441
Hardware and software set up . . . . .	442
Stored procedures workload . . . . .	442
<b>Related publications</b> . . . . .	445
IBM Redbooks . . . . .	445
Other publications . . . . .	445
Online resources . . . . .	446
How to get Redbooks . . . . .	446
Help from IBM . . . . .	446
<b>Index</b> . . . . .	447



# Figures

1-1 Phantom row and non-repeatable read anomalies . . . . .	6
1-2 Dirty read and lost update anomalies . . . . .	7
1-3 Logical units of work and DB2 units of recovery . . . . .	11
2-1 Object locking hierarchy . . . . .	15
2-2 Acquire and release of table space and table locks . . . . .	20
2-3 Duration of page or row locks in a read-only environment . . . . .	21
2-4 Duration of page or row locks in a read and write environment . . . . .	22
2-5 Lock escalation for a partitioned table space . . . . .	31
2-6 Lock suspension and timeout . . . . .	32
2-7 Simple illustration of a deadlock . . . . .	33
3-1 RR readers and concurrent insert process: semantics problem and implementation. .	42
3-2 End-of-file index lock. . . . .	43
3-3 Lock processing . . . . .	47
3-4 IRLM service class definition in Workload Manager . . . . .	49
4-1 APPEND YES option to minimize number of locks on random INSERTs. . . . .	74
4-2 Lost update anomaly caused by releasing locks too soon . . . . .	77
4-3 A hot page and a hot row . . . . .	81
4-4 Updating a summary table . . . . .	83
4-5 Spreading updates in a summary table . . . . .	83
5-1 Changing a processing sequence to reduce lock duration . . . . .	100
5-2 Page level lock duration for isolation CS versus RS/RR. . . . .	104
5-3 Data inconsistency caused by rollback . . . . .	109
5-4 Data inconsistency between index and data page . . . . .	111
5-5 Sample optimistic locking implementation with ROW CHANGE TIMESTAMP. . . . .	115
5-6 Logical units of work and DB2 units of recovery . . . . .	121
5-7 DB2 locking strategies for online applications . . . . .	124
5-8 Data-only locking for index-only access with lock size row/page . . . . .	140
5-9 Using FOR UPDATE OF to acquire a U lock . . . . .	153
5-10 Deadlocks due to unordered updates . . . . .	154
5-11 Lock escalation controls . . . . .	158
6-1 Concurrency of transactions and utilities . . . . .	166
6-2 Example of partition independence. . . . .	173
6-3 Two LOAD jobs execute concurrently with SQL on two partitions of a table space . .	174
6-4 Potential phantom row anomaly during re-read with isolation RS. . . . .	185
6-5 Locking with CURRENTDATA (YES) . . . . .	188
6-6 Best case of lock avoidance using isolation CS with CURRENTDATA(NO) . . . . .	189
8-1 Threads and resources involved in a deadlock. . . . .	245
8-2 Excerpt from DSNESM68 SQL statements . . . . .	251
8-3 List dynamic statement cache. . . . .	252
8-4 SQL statement text . . . . .	252
9-1 List OBID 31 . . . . .	283
9-2 SQL statements of EMPDEL package . . . . .	290
9-3 Statement #365 of the EMPSEL package . . . . .	294
9-4 Complete deadlock picture . . . . .	299
10-1 DB9CG DB2 data sharing group. . . . .	307
10-2 Data sharing global locking flow . . . . .	308
10-3 DB2 data sharing global locking system components . . . . .	309
10-4 Overview of CF structures used by DB2. . . . .	312

10-5	Coupling Facility lock structure . . . . .	313
10-6	Data sharing page set/partition P-lock: two readers . . . . .	317
10-7	Data sharing page set/partition P-lock: inter-DB2 read/write interest . . . . .	319
10-8	Data sharing page set/partition P-lock: inter-DB2 write/write interest . . . . .	321
10-9	Lock optimization example . . . . .	324
10-10	Explicit parent/child hierarchy . . . . .	325
10-11	Data sharing parent L-Lock: inter-DB2 read-read interest . . . . .	327
10-12	L-locks with inter-DB2 read/write interest . . . . .	328
10-13	Page P-lock usage with LOCKSIZE ROW . . . . .	330
10-14	Space map page contention . . . . .	333
10-15	Pseudo-close processing for a single member going through pseudo-close . . . . .	335
10-16	Multiple members losing interest . . . . .	337
10-17	Reader quits first. . . . .	338
10-18	Retained page set or partition P-locks after a member failure . . . . .	352
10-19	False contention resolution . . . . .	355
10-20	XES level resource contention . . . . .	357
10-21	Global lock management . . . . .	358
10-22	Contention management. . . . .	359
10-23	Global deadlock detection. . . . .	362
A-1	DB2 configurations . . . . .	442
A-2	The GLW database . . . . .	443

# Tables

1-1	Anomalies seen at the different isolation levels .....	9
2-1	Compatibility of page and row lock states .....	18
2-2	Compatibility of table space, partition, and table lock states .....	19
2-3	Lock avoidance and cursor access .....	36
2-4	Impact of CURRENTDATA option .....	38
3-1	Deadlock SQL return codes .....	48
3-2	DB2 subsystem locking activity DBD - (d) .....	53
3-3	Claim classes .....	54
3-4	Differences between the latch and the lock .....	58
3-5	Latch class .....	58
3-6	Latch class contention counts: when to pay attention .....	60
4-1	Default LOCKSIZE values for various table space types .....	68
4-2	Possible values of LOCKMAX .....	70
4-3	Default value of LOCKMAX .....	70
4-4	Sequence objects: impact of caching on order .....	80
5-1	Possible lock modes for LOCKSIZE=ANY, PAGE, or ROW for various access paths ..	89
5-2	Possible lock modes for LOCKSIZE=TABLE irrespective of the access path .....	97
5-3	Possible lock modes for LOCKSIZE=TABLESPACE irrespective of the access path ..	98
5-4	Read-only or ambiguous cursor with HOLD and table space lock release behavior ..	99
5-5	Locking and concurrency techniques and isolation levels .....	102
5-6	Modes of locks acquired by LOCK TABLE .....	105
5-7	Sample SQL statements that use mass delete locks .....	112
5-8	Concurrency options versus accuracy .....	119
5-9	Termination of a unit of recovery .....	123
5-10	TRUNCATE statement locking behavior for different table space types .....	130
5-11	Cursor concurrency for each cursor type .....	131
5-12	Sample scenario showing lock interaction between SQL statements .....	133
5-13	Sample scenario showing a INSERT timeout .....	134
5-14	XML locks for various types of SQL statements .....	138
5-15	How to interpret TSLOCKMODE for various table space types and LOCKSIZEs ..	142
5-16	Lock mode as determined by the isolation level at run time .....	142
5-17	Comparison of identical workload with different choice of locking controls .....	148
5-18	Workload SQL statement profile .....	149
5-19	List of common lock types used in OMEGAMON PE V4.2 .....	159
6-1	Claim classes and how DB2 uses them .....	162
6-2	Table space (Part 1) lock mode and claim counts at time=T0 .....	167
6-3	Table space (Part 1) lock mode and claim counts at time=T1 .....	167
6-4	Table space (Part 1) lock mode and claim counts at time=T2 .....	167
6-5	Table space (Part 1) lock mode and claim counts at time=T3 .....	168
6-6	Table space (Part 1) lock mode and claim counts at time=T4 .....	168
6-7	Table space (Part 1) lock mode and claim counts at time=T5 .....	168
6-8	Table space (Part 1) lock mode and claim counts at time=T6 .....	169
6-9	Table space (Part 1) lock mode and claim counts at time=T7 .....	170
6-10	Table space (Part 1) lock mode and claim counts at time=T8 .....	170
6-11	Table space (Part 1) lock mode and claim counts at time=T9 .....	170
6-12	Table space (Part 1) lock mode and claim counts at time=T10 .....	171
6-13	Table space (Part 1) lock mode and claim counts at time=T11 .....	171
6-14	Table space (Part 1) lock mode and claim counts at time=T12 .....	171

6-15	Events associated with concurrently running LOAD utilities along with SQL . . . . .	174
6-16	Locks and drains used by the STOP DATABASE command . . . . .	180
6-17	Default ACQUIRE and RELEASE values for different bind options . . . . .	181
6-18	The default ISOLATION values for different types of BIND operations . . . . .	184
6-19	Bind process interaction with SQL . . . . .	191
6-20	Contention for locks on a DBD in the EDM DBD cache . . . . .	193
6-21	Restricted states . . . . .	194
6-22	Restricted state check during SQL program process . . . . .	199
7-1	IRLM startup procedure options . . . . .	202
7-2	Timeout multiplier and execution environment . . . . .	206
9-1	LOCKINFO information . . . . .	258
10-1	IRLM and XES Locking differences . . . . .	310
10-2	IRLM parent L-lock to XES lock mode mapping . . . . .	310
10-3	IRLM page set P-lock to XES lock mode mapping . . . . .	311
10-4	IRLM child L-lock and page P-lock to XES lock mode mapping . . . . .	311
10-5	Distinguishing features between L-locks and P-locks . . . . .	315
10-6	Page set P-lock modes depending on the DB2 member interest . . . . .	317
10-7	Determining when child locks are propagated to XES . . . . .	326
10-8	Recommendations for the lock structure . . . . .	342
10-9	Lock entry size and corresponding MAXUSRS . . . . .	345
10-10	Retained P-lock modes . . . . .	351
11-1	Locking counters overview . . . . .	377
11-2	Comparison for CD=NO and CD=YES with ISOLATION(CS) . . . . .	393
11-3	Comparison of LOCKSIZE ROW versus LOCKSIZE PAGE MAXROWS 1 . . . . .	395
11-4	High INSERT scenario options . . . . .	399
11-5	High INSERT scenarios comparison . . . . .	400
12-1	Table space GLWSEPA claim and drain classes . . . . .	435
12-2	Claim and drain classes in REORG log phase . . . . .	438
A-1	GLW table profiles . . . . .	443



# Examples

1-1 Example of a unit of work . . . . .	10
4-1 RANDOM sequence index DDL . . . . .	86
4-2 RR reader: sample SQL used. . . . .	86
5-1 Sample INSERT statement. . . . .	92
5-2 Lock information for a static INSERT ... VALUES(...) statement. . . . .	92
5-3 Claimers while performing an INSERT . . . . .	93
5-4 Sample INSERT .... fullselect statement. . . . .	93
5-5 Locks acquired on a INSERT ... fullselect (1 row): OMEGAMON PE output . . . . .	94
5-6 Sample timeout error message on a SELECT with UR with concurrent mass delete. . . . .	113
5-7 Sample DISPLAY DATABASE command output when a mass delete is in progress . . . . .	113
5-8 SKIP LOCKED DATA and gross locks . . . . .	119
5-9 Locks acquired by REFRESH statement on an MQT created with ISOLATION(UR). . . . .	127
5-10 Locks acquired by REFRESH statement on an MQT created with ISOLATION(RR) . . . . .	128
5-11 Locks acquired while an EXCHANGE statement is executed. . . . .	129
5-12 DISPLAY DATABASE command fails when an EXCHANGE statement is running . . . . .	129
5-13 Usage restrictions for TRUNCATE table statement . . . . .	130
5-14 Locks acquired by a sample MERGE statement . . . . .	132
5-15 Locks acquired by a sample SELECT FROM UPDATE statement. . . . .	133
7-1 How to display the IRLM storage above the bar . . . . .	203
7-2 How to dynamically adjust the limit for above the bar private storage . . . . .	203
7-3 How to display the deadlock and timeout value . . . . .	205
7-4 How to modify the timeout value at IRLM level. . . . .	205
7-5 How to modify the deadlock time at the IRLM level . . . . .	207
8-1 Plan level CLASS 3 suspensions in a DB2 accounting report . . . . .	221
8-2 DSNT375I message . . . . .	222
8-3 DSNT376I message . . . . .	223
8-4 DSNT501 message. . . . .	224
8-5 00C900E8 and 00C9008E reason codes . . . . .	224
8-6 DSNIO31I message. . . . .	225
8-7 DSNJ031I message . . . . .	225
8-8 DSNR035I message . . . . .	226
8-9 DSNR036I message . . . . .	226
8-10 Statistics highlights: subsystem services in remote location sections. . . . .	229
8-11 Non-data sharing statistics locking activity . . . . .	230
8-12 DSNT500I message . . . . .	233
8-13 Statistics CPU times . . . . .	233
8-14 DB2 latch contention per second . . . . .	234
8-15 SQL DCL. . . . .	235
8-16 Accounting highlights information . . . . .	236
8-17 Elapsed time distribution. . . . .	237
8-18 Accounting class 1,2,3 times . . . . .	237
8-19 Global contention L-locks and P-locks . . . . .	240
8-20 Accounting locking information . . . . .	241
8-21 Accounting DRAIN/CLAIM information . . . . .	242
8-22 Accounting SQL DCL information. . . . .	242
8-23 IFCID 172 in a LOCKOUT trace . . . . .	243
8-24 Retrieving table OBID information from the catalog . . . . .	244
8-25 IFCID 196 in a LOCKOUT trace . . . . .	246

8-26	Retrieving table OBID information from the catalog - 2 . . . . .	246
8-27	IFCID 196 in a LOCKOUT trace - 2 . . . . .	247
8-28	More than URLGWTH log records . . . . .	248
8-29	More than URCHKTH system checkpoints . . . . .	249
8-30	Holding a read claim for more than LRDRTHLD minutes . . . . .	250
8-31	IFCID 337: lock escalation . . . . .	251
9-1	DISPLAY DATABASE CLAIMERS . . . . .	257
9-2	DISPLAY DATABASE LOCKS . . . . .	258
9-3	-DIS THD(*) output . . . . .	259
9-4	-DIS TH(*) output . . . . .	260
9-5	-DIS THD(*) TYPE(INACTIVE) output . . . . .	260
9-6	Timeout with a system thread involved . . . . .	261
9-7	-DIS THD(*) TYPE(SYSTEM) output . . . . .	261
9-8	Resource unavailable at DROP TABLE time . . . . .	263
9-9	DSNT500I console message with RC 00E70081 . . . . .	263
9-10	Selecting users from an object in the DSC . . . . .	264
9-11	CLASS 3 wait times for statements to be released . . . . .	264
9-12	Locking suspension report . . . . .	266
9-13	Resource and type of suspension . . . . .	267
9-14	Locking resume reasons . . . . .	268
9-15	LOCKING TRACE LEVEL(DETAIL) report . . . . .	271
9-16	RECTRACE TRACE LEVEL(LONG) report . . . . .	273
9-17	Comparing LOCKING and RECTRACE . . . . .	276
9-18	Formatted IFCID 218 record . . . . .	277
9-19	List database and table space name . . . . .	278
9-20	Formatted IFCID 223 record . . . . .	278
9-21	List owner and table name . . . . .	278
9-22	Joblog with SQLCODE -913 . . . . .	281
9-23	Messages in the SYSLOG . . . . .	282
9-24	LOCKOUT trace . . . . .	282
9-25	OMEGAMON PE accounting report by plan name . . . . .	285
9-26	EMPDEL package level accounting info class 7 and 8 . . . . .	286
9-27	EMPDEL package level accounting info class 10 . . . . .	286
9-28	Lock request triggering the deadlock . . . . .	288
9-29	SQL statement triggering the deadlock . . . . .	289
9-30	Acquiring the X-lock on page x'3000EF' . . . . .	291
9-31	RECTRACE of deadlock survivor . . . . .	292
9-32	Start of SELECT statement #365 . . . . .	294
9-33	S-lock request for page x'243' . . . . .	295
9-34	Syncpoint records . . . . .	295
9-35	LOCK trace for GLWBSQLP.31.x'000243' . . . . .	296
9-36	RECTRACE of SQL statement acquiring the X-lock . . . . .	297
9-37	INSERT statement #1024 . . . . .	298
9-38	GLWTEMP table . . . . .	300
9-39	RI relationships of GLWTEMP . . . . .	300
9-40	FK column in GLWTPRJ to GLWTEMP . . . . .	301
9-41	GLWFPRJ2 foreign key relationship . . . . .	301
9-42	Lock requests issued by the DELETE FROM GLWTEMP statement . . . . .	302
10-1	DISPLAY DATABASE ... LOCKS command: two readers . . . . .	318
10-2	-DISPLAY DATABASE LOCKS command: inter-DB2 read/write interest . . . . .	320
10-3	-DISPLAY DATABASE LOCKS command: inter-DB2 write/write interest . . . . .	322
10-4	Lock structure definition . . . . .	341
10-5	DISPLAY GROUP command output . . . . .	343

10-6	IXC582I message in SYSOUT log . . . . .	346
10-7	DISPLAY GROUP command output. . . . .	346
10-8	DISPLAY DATABASE ...LOCKS command: retained. . . . .	350
11-1	DISPLAY DATABASE SPACE() LOCKS ONLY command output . . . . .	366
11-2	DISPLAY DATABASE SPACE() LOCKS ONLY command output 2 . . . . .	367
11-3	DISPLAY BUFFERPOOL command with GBPDEP(Y). . . . .	368
11-4	DISPLAY GROUP command output. . . . .	370
11-5	z/OS D XCF command . . . . .	371
11-6	Layout of the STATISTICS SCOPE(GROUP) report . . . . .	373
11-7	Statistics data sharing locking. . . . .	375
11-8	Statistics LOCKING ACTIVITY section for both members . . . . .	379
11-9	Page P-lock counters in the GBP section . . . . .	381
11-10	Accounting Report SCOPE(GROUP) . . . . .	383
11-11	Accounting report data sharing section. . . . .	384
11-12	Global contention section . . . . .	385
11-13	Accounting report locking section . . . . .	388
11-14	Group buffer pool section . . . . .	388
11-15	RMF coupling facility activity report . . . . .	390
11-16	SQL DML accounting report section: example 1 . . . . .	392
11-17	SQL DML accounting report section: example 2 . . . . .	395
11-18	Global contention section: example 1 . . . . .	396
11-19	Group buffer pool section: example 1 . . . . .	397
11-20	Stored procedures workload output . . . . .	398
11-21	Accounting report with SCOPE(GROUP) example. . . . .	398
11-22	SQL DML accounting report section: example 3 . . . . .	399
11-23	Global contention section: example . . . . .	401
11-24	Group buffer pool P-lock activity . . . . .	401
11-25	Global contention and group buffer pool page P-locks . . . . .	402
11-26	Global contention and group buffer pool P-lock activity . . . . .	403
11-27	Accounting information: plan level information . . . . .	404
11-28	Accounting information: package level information. . . . .	407
11-29	SCOPE(GROUP) lock suspension report. . . . .	408
11-30	Edited lock suspension report. . . . .	409
11-31	Lock suspensions or GLWXEMP3 . . . . .	410
11-32	GLWXEMP3 definition . . . . .	410
11-33	Lock suspension trace for an object . . . . .	412
11-34	Suspended page P-lock request. . . . .	413
11-35	Insert statement . . . . .	414
11-36	P-lock holder on D9C1 . . . . .	415
11-37	INSERT statement issued by D9C2 . . . . .	416
12-1	Isolation UR reader example . . . . .	428
12-2	Utilities and SQL in data sharing: inter-DB2 read/write. . . . .	435
12-3	utilities and SQL in data sharing: inter-DB2 write/write. . . . .	436
12-4	DISPLAY BUFFERPOOL...GBPDEP(YES) output. . . . .	437
12-5	Page set P-locks for concurrent utility access . . . . .	438



# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.


## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

# Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®	OS/390®	Tivoli®
DB2®	Parallel Sysplex®	VTAM®
DRDA®	Redbooks®	WebSphere®
IBM®	Redpaper™	z/OS®
IMS™	Redbooks (logo)  ®	z9®
InfoSphere™	System z9®	
OMEGAMON®	System z®	

The following terms are trademarks of other companies:

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

# Preface

Locking is the generic term used to refer to the database management system function that is required for managing interprocess concurrency and maintaining data integrity. However, locking is just one of the serialization mechanisms available in IBM® DB2® for z/OS®. DB2 uses different mechanisms for serialization to achieve its goal of maximizing concurrency without losing integrity with a minimum cost in CPU, I/O, and storage resources.

In this IBM Redbooks® publication, we review and explore the different serialization mechanisms used in DB2, such as transaction (DML) locking, claims and drains, restrictive states, latching, and optimistic serialization.

This book was written for application developers in order to help them better understand serialization mechanisms and how they influence application design decisions.

## The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

**Paolo Bruni** is a DB2 Information Management Project Leader at the International Technical Support Organization based in Silicon Valley Lab, San Jose, California. In this capacity, he has authored several IBM Redbooks publications on DB2 for z/OS and Data Management tools, and has conducted workshops and seminars worldwide. During Paolo's many years with IBM, in development and in the field, his work has been mostly related to database systems.

**Mike Bracey** is a DB2 for z/OS systems engineer based in the United Kingdom. He has been with IBM for over 26 years and has worked in various capacities as a DB2 branch specialist, DB2 consultant for designing, implementing and tuning applications, and DB2 beta program manager for Europe. He has extensive practical DB2 experience, gained over 20 years, of DB2 application design, data modelling, and performance monitoring and tuning. He is currently a member of the IBM Software Business in Europe, providing technical support for DB2 for z/OS, DB2 Tools, and related topics. His current areas of interest are optimization, buffer pool tuning, and XML.

**Rubina Goolamhussen** is a Senior IT Specialist with IBM Global Technology Services based in Portugal. She holds a Master's degree in Computer Science and Systems Engineering and she has 14 years of experience in the Information Management field within IBM. Her main role is to perform services and give support to IBM Portugal customers in DB2 for z/OS migrations, database administration, DB2 performance tuning, backup and recovery, and problem determination. Before joining IBM in 1995, Rubina worked with Lisbon's University of Science and Technology, performing science investigation and teaching classes.

**Ravikumar Kalyanasundaram** is a certified IT Specialist. He works as a Managing Consultant with IBM SWG. Ravi has over 16 years of experience with database technology. He provides DB2 performance consulting services for large customers on z/OS and Linux®, UNIX®, and Windows® (LUW) systems. He holds a Bachelor's degree in Electrical and Electronics Engineering and a Master's degree in Business Administration. Ravi is also a certified Database Administrator on DB2 V7, V8, and V9. His interests are DB2 performance tuning and disaster recovery.

**Bart Steegmans** is a Senior DB2 Product Support Specialist from IBM Belgium, currently working remotely for the Silicon Valley Laboratory in San Jose, providing technical support for DB2 for z/OS performance problems. Bart was on assignment as a Data Management for z/OS project leader at the ITSO, San Jose Center, from 2001 to 2004. He has over 20 years of experience in DB2. Before joining IBM in 1997, Bart worked as a DB2 system administrator at a banking and insurance group. His areas of expertise include DB2 performance, database administration, and backup and recovery.



*Figure 1 The authors at SVL (left to right): Bart, Ravi, Rubina, Paolo, and Mike*

Thanks to the following people for their contributions to this project:

Rich Conway  
Bob Haimowitz  
Emma Jacobs  
International Technical Support Organization

John Campbell  
Tammie Dang  
David Herlich  
Akiho Hoshikawa  
Jeff Josten  
Karelle Cornwell  
Gopal Krishnan  
Laura Kunioka-Weis  
Qing-ying Li  
Xun Li  
Roger Miller  
Manvendra Mishra  
Ron Parrish  
Emily Prakash  
Akira Shibamiya  
Jim Teng  
John Tobler



Steve Turnbaugh  
Rich Vivenza  
Ping Wang  
Julie Watts  
IBM Silicon Valley Lab, San Jose

Rick Butler  
Bank of Montreal Financial Group, Toronto, Canada

Glenn McGeoch  
IBM USA

Judy Ruby-Brown  
IBM Advanced Technical Support, Dallas

Thanks to the authors of the previous edition of this book, *Locking in DB2 for MVS/ESA Environment*, SG24-4725, published in August 1996:

- ▶ Ravi Kumar
- ▶ Carlos Guardia
- ▶ Hans Ulrik Tetens

## Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an e-mail to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400



# Part 1

## Concurrency and integrity

In this part, we introduce serialization and describe the various techniques that DB2 provides for ensuring application concurrency with data integrity.

This part contains the following chapters:

- ▶ Chapter 1, “Serialization” on page 3
- ▶ Chapter 2, “Transaction locking” on page 13
- ▶ Chapter 3, “Serialization techniques” on page 41





# Serialization

In this chapter, we introduce serialization and discuss the context in which serialization takes place.

The following topics are covered:

- ▶ Introduction
- ▶ Transaction management
- ▶ The reasons for serialization
- ▶ Isolation levels
- ▶ Unit of work and unit of recovery
- ▶ Serialization mechanisms used by DB2

## 1.1 Introduction

Locking and lock management are vital to ensure data integrity whenever multiple concurrent processes require update access to a shared database. The term “locking” refers to the set of serialization techniques used by DB2, such as latches and locks, to ensure the integrity of data in the database. There are four primary categories of processes that a lock management system must handle:

- ▶ Applications using standard data manipulation language, such as INSERT, SELECT, UPDATE, DELETE and MERGE. Applications can be further grouped into transaction and batch based processing, though in recent years the distinction is becoming blurred.
- ▶ Housekeeping operations performed by utilities, such as COPY and REORG.
- ▶ Application management functions, such as BIND.
- ▶ Data structure changes for new or changed objects, such as CREATE or ALTER TABLE.

The traditional approach has been to focus on application locking, as there was time available over night or on weekends to allocate exclusive access to the database for the other three categories. However, with the advent of 24x7 operations and Internet based applications with unpredictable and uncontrollable workloads, the scope has broadened to include concurrent activity in all four categories.

Locking is a database management system (DBMS) function required to allow interprocess concurrency and avoid data integrity problems, that is, avoid exposing uncommitted updates, avoid missing updates, and provide for repeatable read. Two or more independent processes must be prevented from simultaneously accessing and updating the same data.

It is the combination of concurrency, updating, and the need for consistency provided by the DBMS and the application that drive granularity and locking. The job is a partnership. Everything that the DBMS can *provide*, the application can undo with a combination of caching data across commits, failure to check return codes, inadequate and incorrect use of searched updates outside of the cursor, or bad restart logic.

Without proper locking, application processes that update would have to be single-threaded, thereby reducing throughput and increasing response time. Running batch processes concurrently with online processes would be nearly impossible.

But the lock manager must do more than set and release locks. Management of processes suspended because of requests for locked resources is essential. Dispatching these suspended processes when the resource becomes available is a basic task. Suspended processes that represent a deadlock situation must be identified and the deadlock relieved, while the integrity of the database is maintained.

The power of IBM System z® servers combined with the advanced scalability provided by the parallel sysplex feature of z/OS and DB2 data sharing means that workloads of 200,000 SQL statements per second or more are now feasible. These workloads are driven not only through the traditional interfaces of IMS™, DB2, and CICS®, but also through newer applications built in to WebSphere® using Java™ to deliver the latest business requirements. This has resulted in workloads serving many more interactive users with many more concurrent threads, which inevitably increases the likelihood of lock suspensions with the consequent risk of timeouts. As with deadlocks, processes that have timed out must be backed out while maintaining database integrity.

## 1.2 Transaction management

Serialization is a key contributor to the ability of a database management system to achieve the so called Atomicity, Consistency, Isolation, and Durability (ACID) properties of a transaction. From a database perspective, a transaction is a single logical unit of work consisting of a set of SQL statements that make up a business process. The definitions of the ACID properties are:

- ▶ Atomicity

Atomicity is the ability of the database to ensure that either all changes are made or none of them are made. In other words, there can be no partial implementation of the updates within a transaction; it is all or nothing.

- ▶ Consistency

Consistency ensures that the contents of the database remain consistent according to the rules defined, such as referential integrity, both before and after the transaction is executed, whether or not the transaction is successful. Simply put, the database must enforce all consistency rules that have been defined.

- ▶ Isolation

Isolation requires that transactions be executed as though no other transactions are being executed at the same time. Other transactions should not be aware of any intermediate states.

- ▶ Durability

Durability requires that once the transaction is committed and has notified the application of success, then any updates to the database are preserved irrespective of any system failures, such as power failure.

A logical unit of work in an application is a set of SQL statements, such as SELECT, INSERT, UPDATE, MERGE, or DELETE. At any point during the logical unit of work, a failure may require the rollback of all previous updates in that logical unit of work in order to keep the database in a state that is consistent with the business rules. At the end of the logical unit of work, the data manipulated by the business process should be in a state that is consistent with the business rules.

In DB2, a unit of recovery is the set of INSERT, UPDATE, MERGE, or DELETE statements that are performed from one point of consistency to another by a single application process. A unit of recovery begins with the first change to the data after the start of the process or following the last point of consistency and ends at a commit point.

## 1.3 The reasons for serialization

The serialization techniques used by DB2 are primarily concerned with the isolation property of transaction management. This property deals with preventing anomalies that might result from interference among multiple users who are interacting with the database at the same time.

Here are some examples of possible anomalies that might result from a lack of isolation:

► **Phantom row anomaly**

Suppose that you ask your ticket agency to list all the performances of the Metropolitan Opera for May, and you get a list of four performances. You ask for tickets to all of them, but when you receive the tickets, you discover that another performance has been added and you have to pay for five tickets. This is called the phantom row anomaly, because a piece of data has appeared where you were told that no such data existed. See Figure 1-1 for an illustration.

► **Non-repeatable Read**

Suppose that you ask your ticket agency for the price of tickets to a concert and you are told that the price is \$35. You decide to buy some tickets, but only after you have made your purchase you discovered that the price has gone up to \$50. This is called the non-repeatable read anomaly, because you have accessed the same data twice and found different values. See Figure 1-1 for an illustration.

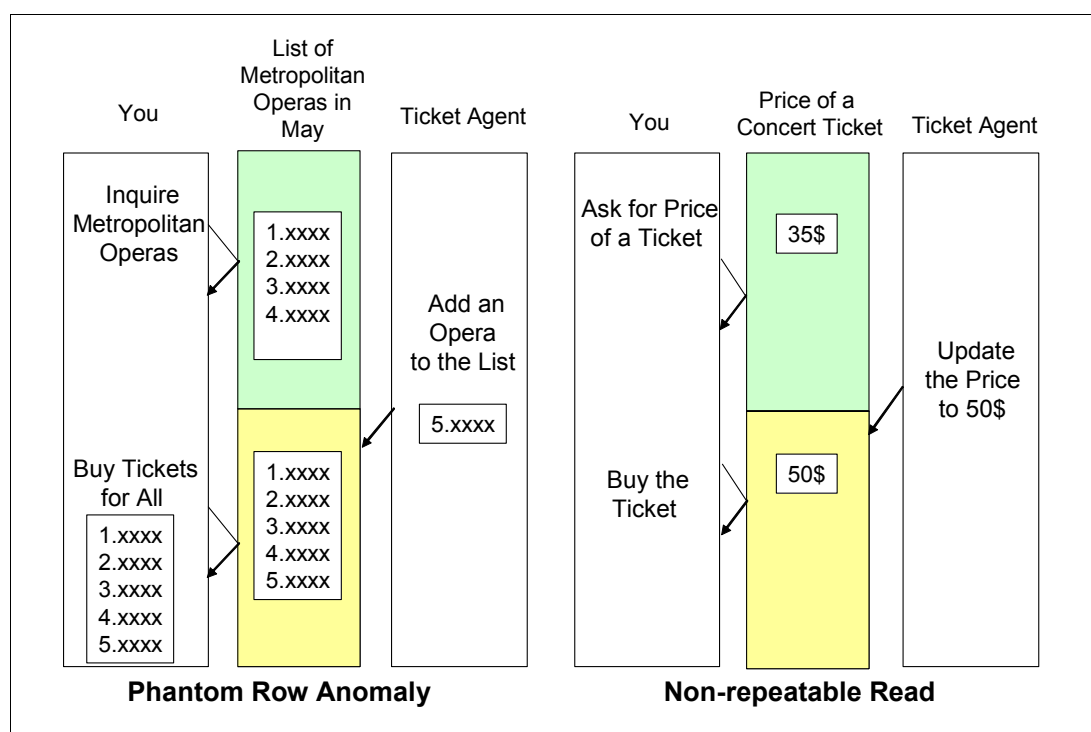


Figure 1-1 Phantom row and non-repeatable read anomalies

► **Dirty Read anomaly**

Suppose that you scan a list of planned concerts and see that Willie Nelson will be performing in your town next summer. But when you try to buy a ticket, you discover that the list was only tentative and that the concert was never really scheduled. This is called the dirty read anomaly, because you were allowed to read information before it was committed. See Figure 1-2 on page 7 for an illustration.



► **Lost Update anomaly**

Suppose that you ask your ticket agency if any tickets are available for a Bruce Springsteen concert, and the agency replies that one ticket is left. A short time later I ask my agency for a ticket for the same concert and get the same reply. Your agency prints a ticket and updates the available tickets from one to zero; then my agency does exactly the same thing without seeing your agency's updates. This is called the lost update anomaly, because two users have updated the same data and one of the updates was lost. See Figure 1-2 for an illustration.

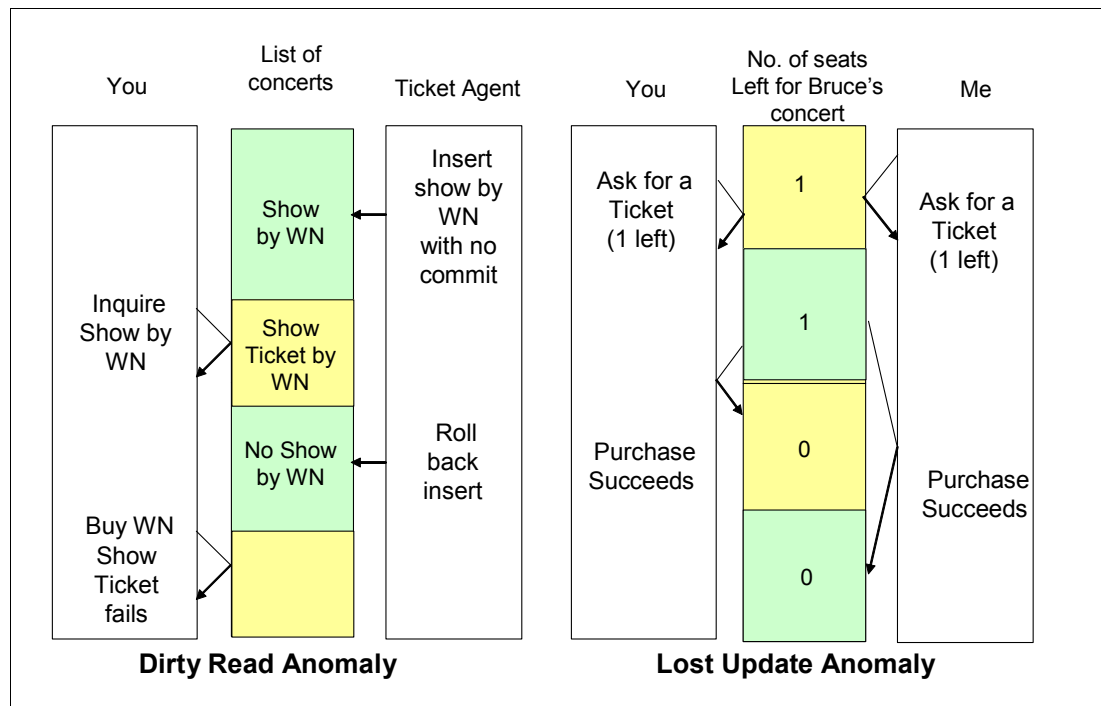


Figure 1-2 Dirty read and lost update anomalies

## 1.4 Isolation levels

Ideally, you would like to avoid all these anomalies. Unfortunately, however, there is a cost associated with this avoidance. The cost is that while you are making up your mind about what tickets to buy, the system must prevent all other users from buying tickets, or updating ticket prices, or changing the concert schedule. This limits the concurrency of the database, which is the ability of the system to provide service to multiple users at the same time. DB2 allows application designers to control the trade-off between isolation and concurrency, by specifying an isolation level for each transaction.

DB2 supports the following four isolation levels:

► **Repeatable Read (RR)**

This is the highest level of isolation, and it prevents all the anomalies described above. If one reads the same piece of data twice in the same transaction with an isolation level of RR, it is certain to see the same set of values (no more and no less). If an RR-level transaction reads a lot of data, the concurrency of the database can be severely limited.

► Read Stability (RS)

This isolation level makes sure that if a transaction reads the same row twice, it will have the same value, but it does not prevent new rows from appearing during the course of a transaction. An RS-level transaction has a smaller impact on concurrency than an RR-level transaction, and it is protected from non-repeatable read anomalies, but not from phantom row anomalies.

► Cursor Stability (CS)

This isolation level locks any row on which the cursor is positioned during a unit of work but keeps it for a minimum time. The lock on the row is held until the next row is fetched or the unit of work is terminated. If a row has been updated, the lock is held until the unit of work is terminated. A unit of work is terminated when either a COMMIT or ROLLBACK statement is executed. It ensures that your application does not read a row that another application process has changed until that process has released that row, but it does allow other application processes to change a row that your application reads before your application commits or terminates.

This isolation level does not allow a row of a table to be changed while your transaction has the cursor positioned on that row. This means that you can read data by fetching from a cursor, then update the current row of that cursor without danger that someone else has updated the row since you read it. If you execute the same query more than once in a CS-level transaction, you may get different answers, but each answer will contain data that was committed at the time you read it. CS-level transactions are protected against dirty reads and lost updates, but not against phantom reads and non-repeatable reads. The CS-level of isolation provides less protection than RR and RS but has much less impact on concurrency. When you want the maximum concurrency while seeing only committed data from concurrent applications, choose this isolation level.

For local access, for read-only and ambiguous cursors<sup>1</sup> in applications bound with ISOLATION(CS), the CURRENTDATA option tells DB2 whether the data upon which your cursor is positioned must remain identical (CURRENTDATA YES) or “current with” the data in the local base table. With CURRENTDATA(YES), DB2 will acquire page or row locks to ensure this data currency. With CURRENTDATA(NO), DB2 will avoid taking locks. ISOLATION(CS) with CURRENTDATA(NO) is the default BIND option in DB2 9.

► Uncommitted Read (UR)

This is the lowest level of isolation and provides the least amount of protection against anomalies. UR-level transactions have virtually no effect on concurrency. Since a UR-level transaction can read data that is in an inconsistent state (for example, it may see the debit to your savings account but not the credit to your checking account), this isolation level is usually used only in statistical surveys or other applications where perfect accuracy is not required.

---

<sup>1</sup> An ambiguous cursor is one for which DB2 cannot determine whether it will be used for update or read-only purposes.

## Anomalies seen at various isolation levels

Table 1-1 summarizes the isolation levels and their protection against the types of anomalies described at beginning of this chapter. A Yes entry indicates that the given anomaly is possible at the given isolation level.

Table 1-1 Anomalies seen at the different isolation levels

Isolation level	Phantom	Non-RR	Dirty Read	Lost Update
Repeatable Read RR	NO	NO	NO	NO
Read Stability RS	YES	NO	NO	NO
Cursor Stability	YES	YES	NO	NO
Uncommitted Read UR	YES	YES	YES	NO

As mentioned, RR prevents all anomalies. RS prevents all except the phantom read anomalies. CS prevents dirty read and lost update anomalies, but does not protect from phantom read and non-repeatable read anomalies.

UR can see all except the lost update anomalies. The lost update anomaly can only occur when both transactions are doing updates. A UR-level read itself will not generate the lost update anomaly. An UR-level update is promoted by DB2 to a CS-level update. Therefore, in effect, an UR transaction will not see the lost update anomalies.

The PREPARE and BIND commands for application programs have a parameter that controls the isolation level for all transactions executed by the program that is being bound. The parameter consists of the word ISOLATION followed by the abbreviation of the desired level: RR, RS, CS, or UR. The SQL SELECT statement can also be issued with a specific ISOLATION level clause. This is discussed in Chapter 5, "Application design" on page 87.

The default isolation level, both for application programs and for interactive sessions, is Cursor Stability (CS) for DB2 9.

## 1.5 Unit of work and unit of recovery

A *unit of work* (UOW) is a sequence of actions that must be completed before any of the individual actions in the sequence can finish. For example, the actions of decrementing an inventory file and incrementing a reorder file by the same quantity constitute a unit of work. Both steps must be successful before the unit of work is complete. If one action occurs and not the other, the database loses its integrity.

The UOW is strictly within the domain of application designers, who must recognize the makeup of each logical application process that changes the content of the database.

Example 1-1 illustrates a DB2 process that consists of one unit of work with two UPDATE statements.

*Example 1-1 Example of a unit of work*

---

```
.....  
UPDATE DEPT-INVENTORY  
SET ONHAND = ONHAND - 144  
WHERE DEPTNUM = 'MACHINING'  
AND PARTNUM = 244925  
.....  
UPDATE DEPT-INVENTORY  
SET ONHAND = ONHAND + 144  
WHERE DEPTNUM = 'ASSEMBLY'  
AND PARTNUM = 244925  
.....
```

---

It does not make sense to remove parts from one area of control in a work-in-process inventory system unless the parts are accounted for as they move to a new area of control. In classic double-entry bookkeeping terms, we cannot credit one account without debiting another.

In this situation, because update operations are involved, the unit of work is also the unit of recovery (UR). That is, a UR is the work done by DB2 for an application that changes DB2 data from one point of consistency to another. A point of consistency (also called a sync point or commit point) is a time when all recoverable data that an application program accesses is consistent with other data. A UR begins with the first change to the data after the beginning of the job (or after the last point of consistency) and ends at the next point of consistency.

The diagram in Figure 1-3 shows the relationship between a DB2 unit of recovery and a logical unit of work in the case of an application using savepoints to allow a rollback of a single logical unit of work.

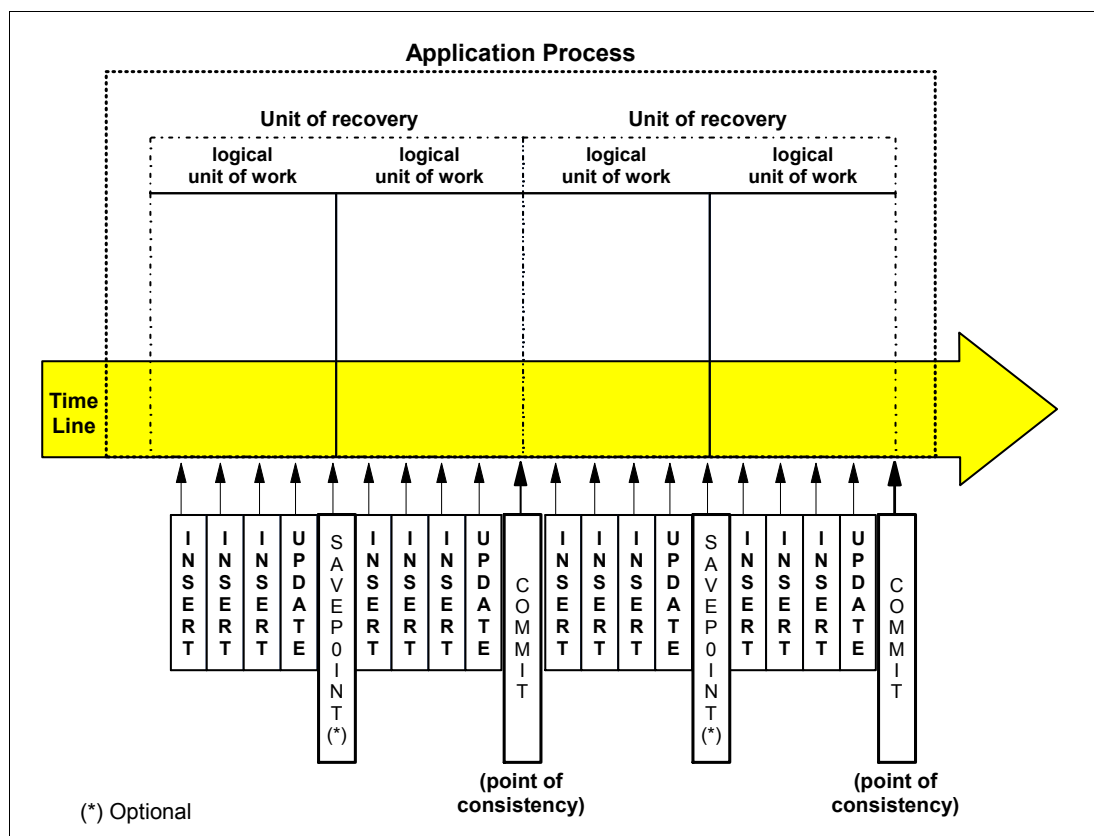


Figure 1-3 Logical units of work and DB2 units of recovery

During a UR, DB2 must enforce strict locking rules to avoid database corruption. URs can end in one of two ways:

- ▶ The application process signals successful completion.
- ▶ The application or the controlling environment indicates an abnormal situation that drives the process back to the beginning of the UR. In other words, an abnormality is removed by backing out any in-flight database changes to the last point of consistency. URs must be 0% or 100% complete; no middle ground is tolerated.

In IMS, the GET UNIQUE from the message queue includes the commit, or the SYNC and CKPT calls can be issued.

In CICS, the commit can be implicit (return) or explicit with a SYNC call.

Distributed, batch, and TSO applications must be designed with commit logic in mind. A commit or sync point generally releases all locks on the DB2 objects. For batch operation, the UOW may consist of one or more transactions if the program is transaction driven or one or more database rows if the program is data driven. For IMS, TSO and CAF batch operations, there is an implicit commit at program termination. Intermediate commit points must be explicitly requested.

The term *logical unit of work* (LUW) is used in transaction managers, such as IMS and CICS, to represent a UOW. In general UOW, LUW, UR, and commit scope are often used as synonymous.

## 1.5.1 Online transaction processing

Online transaction processing typically fits easily into the concept of a logical unit of work. The transaction managers IMS, CICS, and WebSphere coordinate with DB2 for proper synchronization of logging and journaling to ensure a unified commit, and signaling to the lock manager that locks held by a transaction can be released.

Each update transaction represents an LUW and a UR. The transaction boundary represents an implicit UR.

## 1.5.2 Batch and distributed environments

If we think only of online transactions, the term UR seems redundant. When we run DB2 in distributed, batch, or TSO environments, however, we lose the boundaries of the transaction manager responsible for the two-phase commit. If one UOW involves multiple DBMSs, then it may consist of multiple URs. In a distributed environment, a distributed unit of work (DUW) with multi-site updates consists of as many URs as the number of relational DBMSs that participate in the coordinated two-phase commit process.

In distributed, batch, or TSO environments, URs are one or more UOWs that an application process performs against a database between explicit commit points. For TSO, batch, and distributed procedures, the logic for when a UOW has been completed and a UR can be signaled is the application's responsibility. The DB2 COMMIT statement is used with TSO, TSO batch and distributed tasks, the IMS checkpoint (CHKP) call within IMS-controlled batch operation, EXEC CICS SYNCPOINT requests, and the CPIC SRRCMIT function in RRSAP applications. Any of these indicate that the process has completed a so-called durable function that should be reflected in the database, even across system failures.

We have explained locking in simple terms using examples. However, the term *locking* should not be used in a generic sense because locking is just one of the serialization mechanisms employed by DB2. DB2 uses different mechanisms for serialization to achieve its goal of maximizing concurrency without losing the integrity of the database and with a minimum cost in CPU, I/O, and storage resources.

## 1.6 Serialization mechanisms used by DB2

The serialization mechanisms used by DB2 are:

- ▶ Internal Resource Lock Manager (IRLM) managed locks
- ▶ Restrictive states
- ▶ Claims and drains
- ▶ Utility compatibility
- ▶ Latches

The processes that use these mechanisms are:

- ▶ Data Manipulation Language (DML), such as SELECT or INSERT
- ▶ Data Definition Language (DDL), such as CREATE, ALTER, or EXCHANGE
- ▶ Data Control Language (DCL), such as GRANT or REVOKE
- ▶ Utilities, such as COPY and REORG
- ▶ Commands, such as START DATABASE or BIND



# Transaction locking

In this chapter, we describe the methods used by DB2 to avoid the sort of anomalies described in 1.3, “The reasons for serialization” on page 5.

The following topics are covered:

- ▶ Transaction locking
- ▶ Locking control options
- ▶ Lock avoidance

## 2.1 Transaction locking

Transaction locking is of prime interest to application developers and database administrators who need to take into account the potential concurrency, performance, and data integrity issues that can arise from the misuse of locking on DB2 objects. Transaction locking is used whenever application processes access and update data using SELECT, INSERT, UPDATE, MERGE, or DELETE statements. The degree of concurrency of an application is a measure of the number of processes that can run that access and update the same data at the same time without causing unacceptable levels of locks suspensions, timeouts, or deadlocks.

The purpose of a lock is to record the association of a process with an object or resource in DB2. Locks are managed by the Internal Resource Lock Manager (IRLM), which executes in its own address space alongside the system services and database services address spaces. See 3.3, “The Internal Resource Lock Manager (IRLM)” on page 44 for a description. Once granted by IRLM, the process is said to be the “holder” or “owner” of the lock. The objective of the lock management process is to avoid the sort of problem outlined in 1.3, “The reasons for serialization” on page 5 and thereby ensure both the integrity and consistency of the contents of the database.

There are three attributes of a lock that need to be understood in order to design high performance applications:

- ▶ Lock size or scope, such as page or row. This controls how much data is locked.
- ▶ State or mode, such as share (S) or exclusive (X). This controls the degree to which the resource can be accessed concurrently by multiple processes. It is sometimes referred to as the strength of the lock.
- ▶ Duration, such as commit or thread de-allocation. This refers to the length of time that the lock is held.

Each of these attributes, as well as LOB and XML locks, are described in this section.

The locks referred to in this chapter are all logical locks that are owned by an application process and are used for transaction serialization. There are also physical locks that are owned by the sub-system and are used solely to support DB2 data sharing. See Part 4, “Data sharing” on page 303 for a discussion of data sharing locking.

### 2.1.1 Lock size

The size of the lock determines how much data is within the scope of a lock (hence the term *lock size*). The terms *lock scope* and *lock granularity* are synonymous with lock size. For example, a page lock includes all rows on a page, while a row lock just covers a single row. DB2 uses locks of the following sizes:

- ▶ High level locks
  - Table space
  - Partition
  - Table
- ▶ Low level locks
  - Page
  - Row
  - LOB
  - XML



DB2 locks database objects in an implicitly hierarchical fashion, that is, high level locks are obtained first (table space, partition, and table), usually in intent-share (IS) or intent-exclusive (IX) states, and then the locks on the lower level, finer granularity database resources (page and row), are subsequently obtained as they are read or updated.

The implicit hierarchical locking scheme has allowed DB2 to:

- ▶ Support lock escalation (see 2.2.3, “Lock escalation” on page 30)
- ▶ Allow DBAs and application developers to specify the locking granularity at the table space, table, page, or row level

A lower level lock cannot be acquired before the objects above in the hierarchy are locked. See Figure 2-1 for the hierarchy of objects that are locked. For example, in a segmented table space, if a process needs to lock a row in one of the tables, then DB2 must first request locks on the table space and the table. Note that locks are not requested against an index.

Notice that DB2 9 processes the simple table space the same way as a segmented table space.

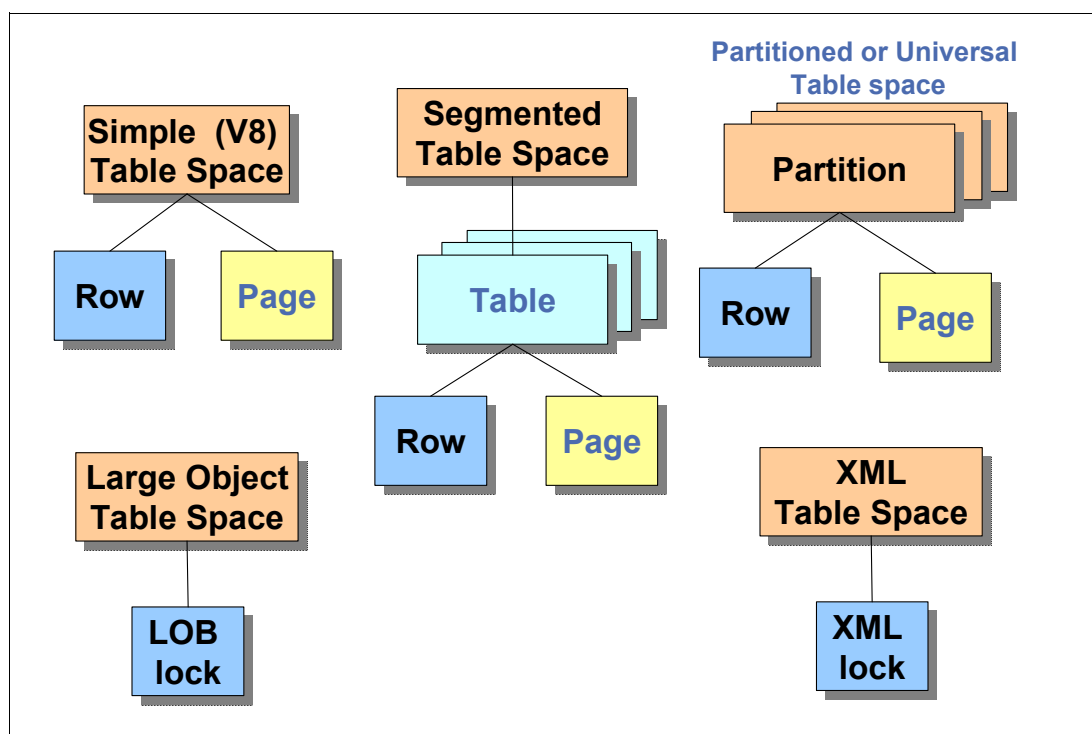


Figure 2-1 Object locking hierarchy

A higher level of lock that corresponds to a DB2 database does not exist. This lock, sometimes referred to as a DBD lock, is primarily used to support the serialization of DB2 structure management when using Data Definition Language. Applications do not generally use this lock; the exception being when dynamic SQL is used with no dynamic statement cache defined (DSNZPARM CACHEDYN=NO), and then DB2 will request a data base lock on all databases touched by the dynamic SQL.

The partition level lock applies to partitioned table spaces, partition-by-growth, and partition-by-range universal table spaces.

The lock size is determined by the LOCKSIZE parameter of the CREATE and ALTER TABLESPACE data definition language statements. The options are ANY, TABLESPACE, TABLE, PAGE, ROW, or LOB or XML. ANY is the default. TABLE should only be used for segmented table spaces. LOB should only be used for LOB table spaces. When ANY is chosen, DB2 uses PAGE, though the manual *DB2 Version 9.1 for z/OS SQL Reference*, SC18-9854 states that DB2 reserves the right to choose ROW.

There is no table space lock for a partitioned table just partition level locks. This means that in the following cases, DB2 will lock all partitions:

- ▶ The PLAN is bound with ACQUIRE(ALLOCATE).
- ▶ The table space is defined with LOCKSIZE TABLESPACE.
- ▶ The LOCK TABLE statement is used without the PART option.

## 2.1.2 Lock state or lock mode

The state of a lock, also called the mode, determines the type of access to the locked object permitted to the lock owner and to any other concurrent application processes. The lock states available for an object depend on the lock size. There are three possible states for a low level lock:

- ▶ Share
- ▶ Update
- ▶ Exclusive

These same lock states are available for the high level lock sizes of table space, partition, or table with the addition of three intent locks referred to as intent share, intent exclusive, and share with intent exclusive. The primary difference is that whenever an intent lock is held, DB2 also has to request a lock at the page or row level, hence the reference to low level locks. The share, update, and exclusive locks at the table space, partition, and table level are also referred to as gross locks because they do not require low level locks to be taken.

The low level locks, that is page, row, or LOB or XML, are acquired when necessary while a process is being executed. When the high level locks, that is, table space, partition, or table, are acquired for static SQL depends on the ACQUIRE BIND option. For ACQUIRE(ALLOCATE), the high level locks are acquired at the first SQL call. For ACQUIRE(USE), the high level locks are acquired at the first actual use of the table. The ACQUIRE bind options have no influence on *when* high level locks are acquired for dynamic SQL, which is always when the object is first accessed.

### Table space, partition, and table lock states for non-LOB or XML data

These are the six possible lock states for a high level lock:

- ▶ Intent share (IS)

The lock owner can read data in the table space, partition, or table, but not change it. The lock owner may acquire an S lock on the row or page containing the row. Concurrent processes may both read and change data.

- ▶ Intent exclusive (IX)

The lock owner and concurrent processes can read and change data in the table space, partition, or table. The lock owner acquires a U lock or S lock on a page or row. When the data is to be changed, the U lock or S lock is promoted to an X lock. The owner can choose to acquire an X lock if the USE AND KEEP EXCLUSIVE LOCKS isolation clause is used on the SELECT statement.

- Share (S)

The lock owner and concurrent processes can read, but not change, data in the table space, partition, or table<sup>1</sup>. There are no S locks on a page or row.

- Update (U)

The lock owner can read, but not change, the locked data. However, the owner can promote the U lock to an X lock and then change the data. Processes concurrent with the U lock can acquire an S lock and read the data, but no concurrent process can acquire a U lock. The lock owner does not need page or row locks.

- Share with intent exclusive (SIX)

The lock owner can read and change data in the table or table space. Concurrent processes can read the data in the table or table space, but not change it. Only when the lock owner changes data is the X lock acquired on a page or row.

- Exclusive (X)

The lock owner may read and change data in the table space or table. Only the concurrent processes using uncommitted read isolation may access the table space or table.

We can classify these six high level states into two subsets:

- The intent states of IS and IX
- The gross states of S, U, and X

The lock state SIX is actually a hybrid state. From a read perspective, it is a gross lock; from a write perspective, it is an intent lock. The SIX lock arises when a process has a gross S lock on the table, such as after executing a `LOCK TABLE table name IN SHARE MODE`. This means that other processes can read the table but not update it. If the process owning the S lock then updates a row in the table, it must acquire an X lock on the page or row but, before doing so, DB2 must acquire an IX lock at the level above to indicate to other processes that the table contains updated data. DB2 could just promote the gross S lock to a gross X lock, but that would be unnecessarily restrictive for any other process wishing to read the table, so the gross S lock is modified to become shared with intent exclusive or SIX.

Any of the gross locks severely limit concurrency. An S lock restricts the entire application population to read-only status. A U lock also restricts concurrent usage to read-only up to the point where the U lock owner exercises the option to promote the U lock to an X lock. Assuming the lock owner can make that promotion without a timeout or deadlock, the object is now exclusively locked for a single thread. Such locking is a little less severe than using only the X-mode from the outset, but is still very restrictive relative to using the implicit intent locks with the low level locks.

## Page and row lock states

There are the three possible lock states for the low level lock size of a page or row. These are applicable only if there is an intent lock on the table space or partition, and also an intent lock on the table if the table space is segmented.

- Share lock

The lock owner and any concurrent process can read, but not change, the locked DB2 object. Other concurrent processes may acquire share or update locks on the DB2 object. Also called an S lock.

---

<sup>1</sup> For partitioned table spaces, if a single partition is locked, other partitions can be accessed depending upon how they are locked.

► Update

The lock owner can read the DB2 object and has the intention to change it. Concurrent processes may acquire share locks and read the DB2 object, but no other process can acquire an update lock. Update locks are promoted to exclusive locks before DB2 actually changes the DB2 object. Promotion to exclusive lock may cause a suspension if other processes are holding share locks. Also called a U lock.

► Exclusive

Only the lock owner can read or change the locked data, with the following exceptions. Concurrent applications using uncommitted read isolation can read the exclusively locked data. Also, if lock avoidance techniques indicate that the exclusively locked data is committed, that data can be returned to concurrent cursor-stability applications that do not require currency of data (refer to 2.3, “Lock avoidance” on page 33).

### Compatibility rules for lock states

The major effect of the lock state is to determine whether one lock state is compatible with another. Locks of some states do not exclude all other users. Assume that process A holds a lock on a table space that process B also wants to access. DB2 requests, on behalf of process B, a lock of some particular state. If the lock mode of process A permits a request from process B, then the two lock states are said to be compatible.

If the two locks are not compatible, process B cannot proceed. It must wait until process A releases the lock.

### Compatibility of page and row lock states

Compatibility for low level locks is simple to define. Table 2-1 shows whether page or row locks of any two states are compatible (Yes) or not (No). No question of compatibility of page lock with a row lock can arise, because a table space cannot use both page and row locks.

Table 2-1 Compatibility of page and row lock states

Held lock	Requested lock		
	Share	Update	Exclusive
Share	YES	YES	NO
Update	YES	NO	NO
Exclusive	NO	NO	NO

Two or more processes can concurrently access data by using a page or row share lock (S lock). However, two or more processes cannot concurrently access data using a page or row update-intent lock (U lock). That is, only one process can acquire a U lock on a specific page or row at any instant. Even though a page or row may be U locked, other processes can access that page or row (but for read-only) by acquiring an S lock. If the process holding the U lock wants to do an update or delete, the U lock must be promoted to X lock before this can be done.

However, the promotion to an X lock does not occur if any other concurrent process holds an S lock on that page or row. Eventually, assuming the S locks are released before the timeout interval, the U lock is promoted to an X lock. While a page or row is U locked, if any other concurrent process wants to do an update or delete, it must first acquire a U or X lock, but has to wait and eventually either the U or X lock is acquired or a timeout occurs.

### **Compatibility of table space, partition, and table lock states**

Compatibility for table space, partition, and table locks is slightly more complex. Table 2-2 shows whether or not the high level locks of table space, partition, or table of any two modes are compatible.

Table 2-2 Compatibility of table space, partition, and table lock states

	Requested lock					
	Intent share	Intent exclusive	Share	Update	Share with intent exclusive	Exclusive
Held lock	IS	IX	S	U	SIX	X
IS	Yes	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No	No
S	Yes	No	Yes	Yes	No	No
U	Yes	No	Yes	No	No	No
SIX	Yes	No	No	No	No	No
X	No	No	No	No	No	No

### **2.1.3 Lock duration**

The duration of a lock is defined as the length of time a lock is held and varies according to the type of lock. Page or row locks are acquired when individual rows are accessed and are released at different times, depending on whether the data is accessed for read-only or for update and on the isolation level specified. See 1.4, “Isolation levels” on page 7 for a description of the four isolation levels supported by DB2. High level locks are typically acquired implicitly, based on options specified at BIND or REBIND time.

Because DB2 does implicit high level locking, it is critical that the application designer understand the options that influence when high level locks are acquired, when they are released, and the state of the lock. See 5.1, “SQL design” on page 88 for a detailed discussion of the design considerations.

#### **Duration of table space, partition, and table locks**

Every plan executed in DB2 must acquire a high level lock that can either be a table space or partition lock and, if the table space is segmented, then a table lock as well. The exception to this rule is that locks are not acquired on the table space or table if uncommitted read isolation is specified. However, DB2 does acquire a mass delete lock. A mass delete is when a DELETE statement is executed with no WHERE clause such that all rows are deleted. DB2 uses the mass delete lock to serialize a mass delete process with an uncommitted reader.

DB2 provides options to control the point at which the high level locks are requested. Figure 2-2 illustrates the effect of the BIND and REBIND options ACQUIRE and RELEASE on when high level locks are acquired and released. Note that the bind options do not have an effect on high level locks on LOB or XML table spaces.

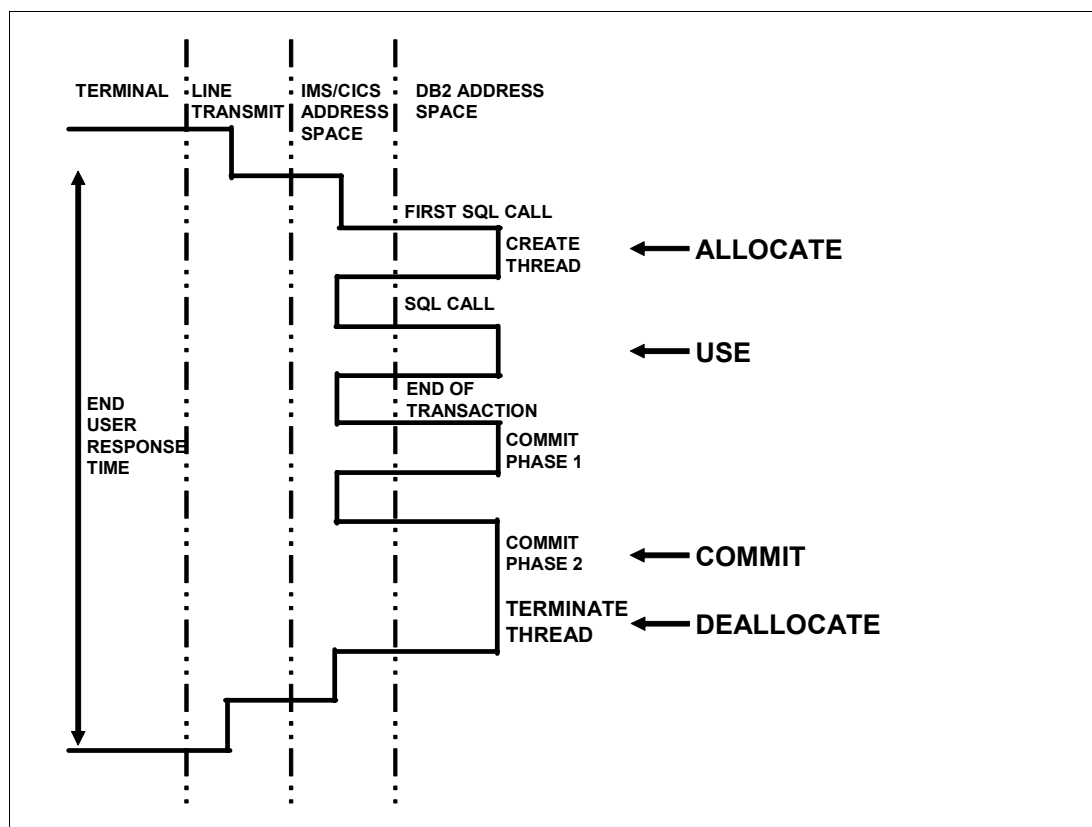


Figure 2-2 Acquire and release of table space and table locks

The possible values for the ACQUIRE option are ALLOCATE and USE. If you use ACQUIRE(ALLOCATE), all of the needed locks on all of the table spaces, partitions, and tables used by the plan are acquired at the first SQL call. If you use ACQUIRE(USE), only the needed locks on the table spaces, partitions, and tables are acquired for a shorter time, as and when the objects are accessed.

DB2 packages and dynamic SQL always use ACQUIRE(USE).

The time when the high level locks are released is influenced by the RELEASE option. The possible values for this option are DEALLOCATE and COMMIT. If you use RELEASE(DEALLOCATE), the locks are released only when the thread terminates. If you use RELEASE(COMMIT), the locks are released at commit time unless there are held cursors.

DRDA® always uses RELEASE(COMMIT). The RELEASE value also has no effect on packages that are executed on a DB2 server through a DRDA connection with the client system.

### Duration of page or row locks

The duration of page or row locks depends on whether they are acquired and released in a read-only or read and write environment.

## Read-only

Figure 2-3 illustrates the effect of the BIND and REBIND ISOLATION option on when page or row locks are acquired and released in a read-only environment.

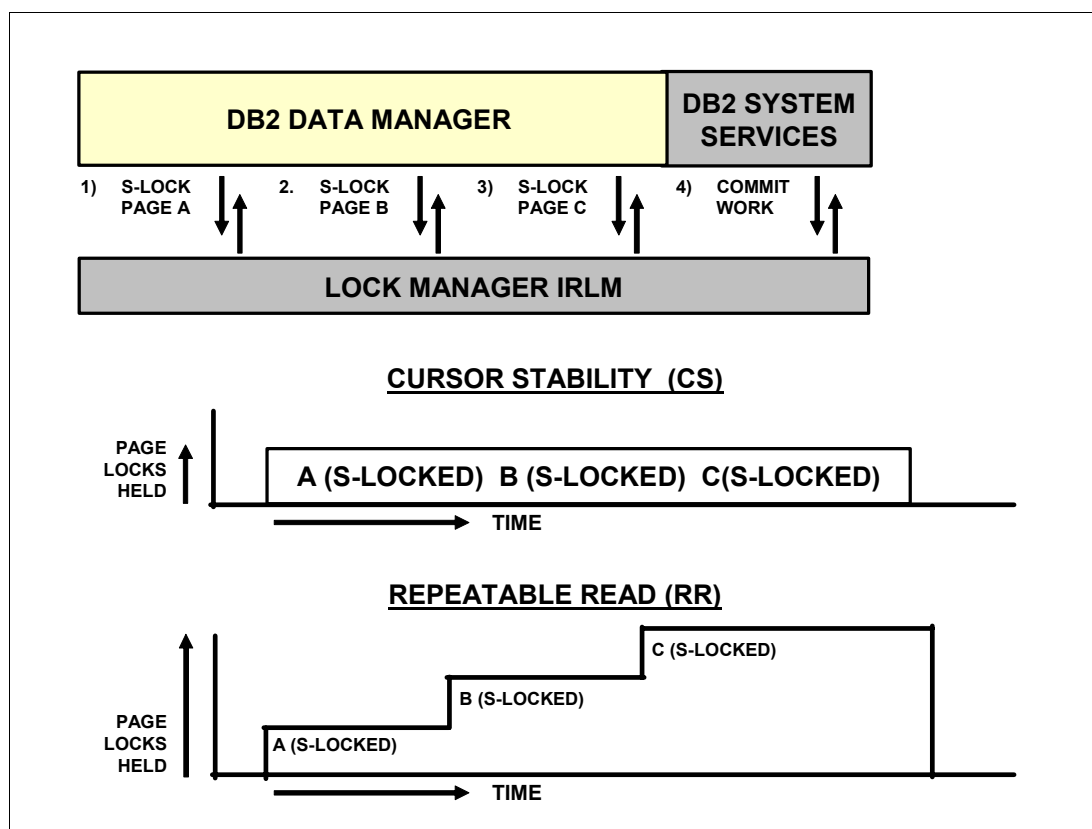


Figure 2-3 Duration of page or row locks in a read-only environment

The duration and number of page or row locks depend on the nature of the process and the value for the ISOLATION option specified at BIND or REBIND. The most common ISOLATION option is cursor stability (CS), which is the default as of DB2 9 for z/OS. With cursor stability as the ISOLATION option, DB2 returns committed data and provides stability of data under updateable cursors. If a query is answered through a work file, for example, a concurrent thread may update the data in the database that generated the data in the work file.

The repeatable read (RR) isolation retains page locks even though the process logic moves through several pages in the same table during a unit of work. If the pages are accessed again, repeatable read provides repeatability: the data cannot be changed by any other concurrent process. This is true for read stability isolation also.

A nonstandard query to list all employees who make more than the average salary is an example of a process that would benefit from repeatable read. Two passes of the table are required: the first calculates the average salary while the second tests the average against each employee's current salary. If employees could be added or deleted, or if salaries could be updated by other concurrent processes, a decision is needed. Either lock out all the concurrent updating processes until the query is complete (repeatable read on the query would do that), or use cursor stability on the query, permitting concurrent access to the table with the knowledge that the query output may contain zero to many erroneous rows. Read stability (RS) would not be enough here, as it would not prevent new employees from being inserted.

If the option WITH HOLD is used in a SQL DECLARE CURSOR statement, *not* all locks are released at commit time:

- All page or row locks are released in DB2 9. In DB2 V8, the page or row lock at the current cursor position is retained (S-lock) or downgraded (X or U --> S) if the DSNZPARM RELEASE LOCKS (DSN6SPRM RELCURHL) option is set to NO. The DSNZPARM default in DB2 V8 is YES. This DSNZPARM has been removed in DB2 9.
- Table space, table, and DBD locks are not released even if RELEASE(COMMIT) is specified.

### Read and write

Figure 2-4 illustrates the effect of the BIND and REBIND ISOLATION option on when page or row locks are acquired and released in a read and write environment.

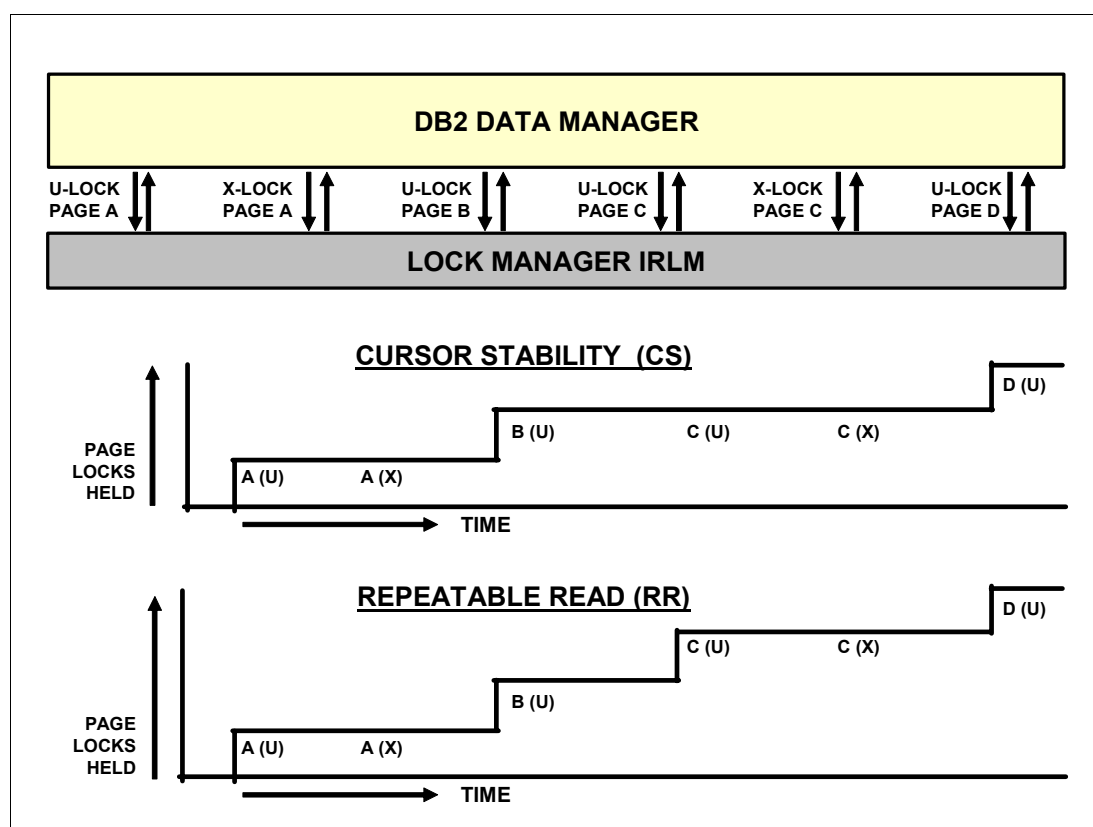


Figure 2-4 Duration of page or row locks in a read and write environment

A process that uses a cursor adds another dimension to locking. Pages read acquire a U-lock, initially indicating update intent. But before a column can be updated or a row can be deleted, the U-lock on the page is changed to an X-lock and is released at commit.



## 2.1.4 Single and two-phase commit write considerations

Programs running in IMS and CICS environments use what is known as two-phase commit. In these environments, the commit process is controlled by the coordinator (IMS or CICS) and DB2 is the participant in the process. Thus, the coordinator requests that the participant prepares to commit and waits for the answer. If the answer is positive, the coordinator requests the commit and the participant commits the changes. If CICS determines that all data being updated is in DB2, then it can switch to a single phase commit, as only the DB2 log is required for recovery.

This process involves writes to the DB2 log. Due to the critical nature of the log for UOW backout and recovery purposes, the standard recommendation is to use dual logs. If dual logs are used, the writes normally occur in parallel, though both must be completed before the locks can be released.

Locks are released when all changes made to the database are secured; that is, the writes to the log containing the changes are completed. Therefore, the time needed for DB2 to write the log data sets is critical to the concurrency of the system; faster writes release locks sooner, decreasing the risk of suspensions. DB2 writes log records in such a way that a set of log writes may include multiple transactions and this, when combined with modern disk technology, can lead to very high rates of commit frequency, such as 5,000 commits per second.

In two-phase commit environments, the commit time is increased by the fact that four writes to the log must be completed (corresponding to Phase 1 and Phase 2 and assuming dual logs). The two subsystems (coordinator and participant) have a message communications delay that must also be taken into account. The writes to the log are parallel when log control intervals are written for the first time and serial when the log control intervals are rewritten. All these facts must be considered when planning the system, as logging time can be a concurrency limiting factor in situations where massive concurrent INSERTs occur. Techniques to improve logging performance include larger OUTPUT BUFFER, faster log devices, preformatted logs, data compression, and data striping. The use of NOT LOGGED can reduce the CPU and elapsed time; however, our experience suggests that the improvement is small.

## 2.1.5 Dynamic statement concurrency

In this section, we focus on those aspects of concurrency that are directly related to dynamic SQL.

### DBD locks

Without statement caching, dynamic DML SQL statements acquire an S-lock on the database descriptor (DBD), when the statement is prepared. This prevents DDL from executing in the database that has been referenced by the dynamic DML statement. The lock is held until the thread reaches a commit point.

When using global statement caching (CACHEDYN=YES), no DBD lock is acquired for short prepares (retrieving a statement from the cache) or for full prepares (inserts into the cache) for SELECT, INSERT, UPDATE, MERGE, or DELETE statements, provided statement caching has not been prohibited for another reason. Those other reasons include the use of the REOPT(ALWAYS) bind option, or if DDL was performed by the application process in the same unit of work. TRUNCATE and EXCHANGE both take a DBD lock.

## Gross lock behavior in dynamic SQL

The following sections describe the management of gross locks (S, SIX, or X) acquired by cached dynamic statements.

Like other packages or plans, prepared dynamic SQL statements may acquire locks on table spaces. When regular plans or packages are bound, the RELEASE parameter specifies when these gross locks will be released:

- ▶ On commit
- ▶ On plan deallocation

Unlike regular plans or packages, dynamic SQL statements are not prepared with such parameters. If a program executes dynamic SQL statements, the objects they lock are locked when first accessed and released at the next commit point with one exception: When plans use RELEASE(DEALLOCATE) and KEEP DYNAMIC(YES), and the subsystem is enabled for dynamic statement caching, the RELEASE(DEALLOCATE) option is honored for dynamic SELECT, INSERT, UPDATE, MERGE and DELETE statements.

Locks acquired for dynamic statements are held until one of the following events occurs:

- ▶ The application process ends (deallocation).
- ▶ The application issues a PREPARE statement with the same statement identifier.  
Locks are released at the next commit point.
- ▶ The statement is removed from the cache because it has not been used.  
Locks are released at the next commit point.
- ▶ An object that the statement is dependent on is dropped or altered, or a privilege needed by the statement is revoked.  
Locks are released at the next commit point.

## Gross lock demotion

DB2 will demote a gross table space, partition, or table lock (a lock with a lock state of S, SIX, or X) to an intent lock (a lock state of IS or IX) at commit for locks held for statements in the dynamic statement cache.

There are several ways in which a gross lock may be acquired on a table space, partition, or a table in a segmented table space. These include:

- ▶ The use of LOCKSIZE TABLESPACE on CREATE or ALTER TABLESPACE
- ▶ The use of LOCKSIZE TABLE on CREATE or ALTER TABLESPACE
- ▶ The use of a table space scan with ISOLATION(RR)
- ▶ The occurrence of lock escalation
- ▶ The selection of a row from a table after the table space has been started for ACCESS(RO)
- ▶ The LOCK TABLE statement
- ▶ Mass delete

For some of these cases, we want to demote the lock state at commit, for others, we do not.

Conditions that prohibit lock state demotion include the following:

- ▶ The table space has a locksize of TABLESPACE.
- ▶ The table space has a locksize of TABLE.
- ▶ The gross lock was acquired as a result of ISOLATION(RR).
- ▶ The gross lock was acquired because the table space was started for ACCESS(RO).
- ▶ The lock is held for static SQL.

In order to demote the lock state of a table or table space lock at commit, all of the following must be true:

- ▶ Statement caching is enabled for the system.
- ▶ KEEP DYNAMIC was specified as a bind parameter.
- ▶ RELEASE(DEALLOCATE) was specified as a bind parameter.
- ▶ The lock is held in a lock state of S, SIX, or X.
- ▶ The lock is acquired for a cached dynamic statement.
- ▶ None of the events that would prohibit lock state demotion have occurred for this agent up to the current commit.

### Selective partition locking

For partitioned table spaces, lock demotion occurs at the table space level, not at the partition level. DB2 does not track eligibility for lock state demotion at a partition level, only at a table space and table level.

For example, suppose there is a dynamic statement that inserts into partition 5 and causes locks to escalate. At commit, the lock is demoted from X to IX. Now suppose that the insert is executed, the lock escalates on partition 5, and then partition 2 is S-locked on behalf of a repeatable read (RR) table space scan. Because we choose not to demote in the case of RR scans, and we only track demotability at the table space level, neither gross lock is demoted at commit.

## 2.1.6 LOB locks

DB2 Large Object (LOB) support was first introduced in DB2 for z/OS Version 6. Since then, there have been a number of improvements to both concurrency and performance culminating in a new locking protocol introduced in DB2 9 for z/OS. The previous locking protocol and the improvements introduced in DB2 9 are fully documented in *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-7270. Provided APAR PK62027 (PTF UK38906) is applied, the improved locking process for LOBs is available in DB2 9 new function mode (NFM) with no requirement for a group wide outage for data sharing.

There are two lock sizes that can be chosen for a LOB table space that are LOCKSIZE LOB and LOCKSIZE TABLESPACE. Choosing LOCKSIZE ANY results in the use of LOCKSIZE LOB.

DB2 9 uses the locks on the base table to provide consistency of the LOBs being accessed by an application. DB2 ensures data integrity by comparing the Read Log Sequence Number (Read LSN or RLSN), from the oldest reader in the system, to make sure that no deleted LOB data pages are reused with new LOB values before the last reader of this particular value has been removed.

A short summary of how locks are established is as follows:

- ▶ INSERT operations require an X-lock on each LOB to be inserted with a manual duration instead of a commit duration (lock and unlock sequence).
- ▶ UPDATE operations require an X-lock on each LOB data to be updated but with a manual duration instead of commit duration.
- ▶ DELETE operations require no LOB lock but an X-lock on the base table row or page.
- ▶ SELECT operations require no LOB lock while accessing the LOB data.
- ▶ UR readers request an unconditional S-LOB lock on each LOB data that is to be selected. This is to serialize with concurrent INSERT or UPDATE operations. The lock is released immediately.
- ▶ Under INSERT or UPDATE operations, DB2 now depends on the Read LSN to decide the availability of each de-allocated page, so no LOB lock is required.
- ▶ In data sharing, DB2 needs to flush LOB data and index pages to GBP before releasing the LOB lock.

See 5.11.1, “LOB locks” on page 135 for a discussion about LOB lock options in your application.

### 2.1.7 XML locks

DB2 9 for z/OS uses a locking scheme for concurrency on XML data to achieve document level consistency and row level consistency. To achieve document level consistency, an XML lock is introduced, which is a lock on an XML document. The same lock modes and compatibility matrix used for the row and page locks are used for XML locks. See 5.11.2, “XML locks” on page 138 for the relationship between XML locks and SQL statements.

### 2.1.8 Global temporary tables

Locking does not apply at all for a created temporary table. For a global declared temporary table (GDTT), no row or table locks are acquired, although share level locks are acquired on the table space and DBD lock on the AS WORKFILE database. A segmented table lock is also acquired when all the rows are deleted from a table or a table is dropped. Locks acquired when declaring or processing GDTTs do not affect concurrency. You cannot issue a LOCK TABLE statement against a declared global temporary table.

### 2.1.9 Global transactions

Global transaction support provides special lock processing functions for the situation where there are multiple DB2 units of recovery that are actually all part of one large, local or distributed, unit of work that is called a global transaction. In particular, locks can be shared between multiple DB2 units of recovery in a single DB2 subsystem when they all belong to the same global transaction.

There are a number of situations in which you may want to combine two or more existing transactions into a single larger transaction. A frequent example occurs in the course of putting an object-oriented interface in front of existing transactions.

Global transaction support in DB2 provides a way to avoid deadlocks on DB2 resources that would otherwise occur when certain kinds of transactions are combined in this way.

This support is not always in effect every time DB2 performs work that is part of a distributed unit of work. It only comes into play when an optional new token (called a global transaction ID (XID)) is supplied to DB2, either through DDF from another DB2 for z/OS subsystem, or through one of the attach facilities that includes the ability to specify the token.

This support is currently provided for DB2 units of recovery that come into DB2 using the IMS attach facility, or the Recoverable Resource manager Services Attachment Facility (RRSAF), or using DRDA through DDF.

The RRSAF support is documented in the DB2 manuals.

## 2.2 Locking control options

You can influence DB2 locking after tables have been created, by using the different options when binding or rebinding an application plan or package or with individual SQL statements.

The options at bind or rebind of the application plan or package are:

- ▶ ACQUIRE, with possible values USE and ALLOCATE
- ▶ RELEASE, with possible values COMMIT and DEALLOCATE
- ▶ ISOLATION, with possible values CS, RR, UR, and RS
- ▶ CURRENTDATA, with possible values YES and NO

The options with SQL statement are:

- ▶ SELECT... WITH CS, RR, UR, or RS
- ▶ INSERT... SELECT... WITH CS, RR, or RS
- ▶ UPDATE... WITH CS, RR, or RS
- ▶ DELETE... WITH CS, RR, or RS
- ▶ DECLARE CURSOR... WITH HOLD
- ▶ DECLARE CURSOR... FOR UPDATE OF
- ▶ LOCK TABLE

See 5.1, “SQL design” on page 88 for a discussion of the locking control options available to an application developer and 6.7, “Locks on DB2 objects” on page 190.

### 2.2.1 Skip locked data

The SKIP LOCKED DATA option allows a transaction to skip rows or pages that are incompatibly locked by other transactions. Because the SKIP LOCKED DATA option skips locked data, the performance of some applications can be improved by eliminating lock wait time. However, you should use the SKIP LOCKED DATA option only for applications that can tolerate the absence of the skipped rows in the returned data. If your transaction uses the SKIP LOCKED DATA option, it does not read or modify data for which other concurrent application processes hold incompatible locks.

DB2 employs various lock avoidance techniques to try to establish the “committedness” of data without resorting to requesting a lock on the data. If those lock avoidance techniques fail, or if locks are required to provide the stability of data required by the isolation level selected, then DB2 is in a position to have to lock data pages or rows. If such locking is detrimental to concurrency requirements, however, even they can be avoided by the use of the SKIP LOCKED DATA clause. This option specifies that a weak semantic is acceptable to enhance concurrency. Note that this option should not be viewed as saying that all data that is locked will be skipped, but that it is acceptable for data that is incompatibly locked to be skipped. To the extent to which locked data can be processed without lock acquisition (lock avoidance techniques successful and no isolation requirement for locking), data will not be skipped.

An example of the use of this option would be an application that uses a DB2 table to control the flow of a process through a number of different states, such as a bank loan approval process. Some application processes will be adding new loan requests while others will be looking for requests that need some action, such as a credit check. The credit check process will run continuously, looking for loan requests in a state requiring a credit check. If some of the loan requests are currently locked, then they will be processed the next time the check process is initiated to look for work to do.

Another example would be when a DB2 table is used as a queue mechanism so that some processes are adding rows to the table while others are removing rows. In this case, it would usually be preferable to use the appropriate technology such as IBM WebSphere MQ for z/OS.

**Important:** When DB2 skips data because of the SKIP LOCKED DATA option, it does not issue a warning. Even if only a subset of the data that satisfies a query is returned or modified, the transaction completes as though no data was skipped. To use the SKIP LOCKED DATA option, you must be willing to accept inconsistent results.

**Recommendation:** For applications and transactions that require fast results and data that may be incomplete, consider using SKIP LOCKED DATA to increase concurrency. For applications and transactions that require finding *all* qualifying rows, do not use the SKIP LOCKED DATA option.

To use the SKIP LOCKED DATA option, specify the clause in a SELECT, SELECT INTO, PREPARE, searched UPDATE, or searched DELETE statement. You can also use the SKIP LOCKED DATA option with the UNLOAD utility.

### Isolation levels for SKIP LOCKED DATA

You can use the SKIP LOCKED DATA option with cursor stability (CS) isolation and read stability (RS) isolation. However, you cannot use SKIP LOCKED DATA with uncommitted read (UR) or repeatable read (RR) isolation levels. For UR and RR, DB2 ignores the SKIP LOCKED DATA clause.

### Lock sizes for SKIP LOCKED DATA

The SKIP LOCKED DATA option works only with the low level row and page locks.

#### ***LOCKSIZE ROW***

All rows with incompatible locks are skipped.

### ***LOCKSIZE PAGE***

All rows on pages with an incompatible lock are skipped. However, if ISOLATION CS is used with CURRENTDATA(NO), then DB2 will be able to access the incompatibly locked page, and look for qualified rows that are committed using lock-avoidance technique without requesting the lock. See 2.3, “Lock avoidance” on page 33 for details.

This can lead to the interesting result where a set of rows can be read with a SELECT as committed data, but a subsequent UPDATE of the same set of rows with SKIPPED LOCKED will result in no updates.

### ***LOCKSIZE TABLE or TABLESPACE***

SKIP LOCKED DATA is ignored.

When locks escalate to a gross lock on table or table space, SKIP LOCKED DATA is ignored.

### ***LOB or XML***

The SKIP LOCKED DATA clause does not apply to LOB or XML locks.

### **Lock mode compatibility for SKIP LOCKED DATA**

Lock mode compatibility for transactions that use the SKIP LOCKED DATA option is the same as lock mode compatibility for other page-level and row-level locks. However, when incompatible locks are held, a transaction that uses the SKIP LOCKED DATA option does not wait for the locks to be released and ignores the locked data instead.

### **Examples of SKIP LOCKED DATA**

We show three examples of the usage of the SKIP LOCKED DATA option.

#### ***Example 1***

Suppose that application process A opens a cursor with the FOR UPDATE clause on a table with page-level locking and that the cursor performs an index scan with data access. Now suppose process B holds an X lock on a page that process A needs to access. Process A is now suspended by IRLM until process B commits and releases the X lock. Note that the two processes may be accessing different rows but a conflict occurs because of page level locking. If process A had used the SKIP LOCKED DATA clause, then this page would have been skipped and process A would have continued to the next data page with a qualifying row in the index, and this assumes that it is acceptable for the type of application.

#### ***Example 2***

Suppose that application process A holds an S lock on a row that process B also wants to access. The query in process B specifies SKIP LOCKED DATA. The outcome of process B depends on the mode of lock that it acquires. If process B requires a compatible S or U lock, process B can access the row on which application A holds the S lock. If process B requires an incompatible X lock, process B cannot access the locked row. Because the SKIP LOCKED DATA option is specified, that row is skipped and the results of process B are returned without that row.

#### ***Example 3***

With some queries that use the SKIP LOCKED DATA clause, you can receive unexpected or inconsistent results and the result may be access path dependent.

Suppose that a table EXTABLE exists in a table space with row-level locking, or in a table space with page-level locking, and the rows of the table are distributed across several pages. EXTABLE has two columns, C1 and C2, that contain the following data:

C1	C2
1	AAAA
2	BBBB
3	CCCC
4	DDDD

Next, suppose that a transaction issues the following update statement:

```
UPDATE EXTABLE SET C1 = 99 WHERE C1 < 3;
```

Suppose that a second transaction issues the following SELECT statement before the previous transaction commits:

```
SELECT COUNT (*) FROM EXTABLE WHERE C2>='AAAA' SKIP LOCKED DATA;
```

If you do not have an index defined on C2, DB2 returns a count of 2 because DB2 skips the two rows that are locked by the uncommitted UPDATE statement. If there exists an index on C2, then DB2 may choose to perform the count with index only access, in which case the answer would be 3, assuming that index lock avoidance took place.

## 2.2.2 Lock promotion

Lock promotion is the action of exchanging one lock on a resource for a more restrictive lock on the same resource, held by the same application process. One example would be an application process that initially acquires an IS lock and then decides to update some of the rows in the table space, which requires an IX lock on the table space. The application is said to promote the table space lock from mode IS to mode IX.

When promoting the lock, DB2 first waits until any incompatible locks held by other processes are released. When locks are promoted, they are promoted in the direction of increasing control over resources: from IS to IX, S, or X; from IX to SIX or X; from S to X; from U to X; and from SIX to X.

## 2.2.3 Lock escalation

DB2 provides the application developer with the option to dynamically switch from low level locking to high level locking based on the number of low level locks held by the application process. This option is called *lock escalation* and is controlled by the parameter LOCKMAX of CREATE TABLESPACE and ALTER TABLESPACE (see 4.1.4, “LOCKMAX” on page 69). The purpose of lock escalation is to reduce the impact of low level locks, both in terms of CPU and storage, at the possible expense of concurrency. When the number of low level locks exceeds a specified threshold, DB2 attempts to promote the high level intent locks, either IS or IX, to S or X, respectively. If this succeeds, then the low level locks are all released and no more low level locks are requested. The lock promotion request may itself fail if other processes hold incompatible locks, either low or high level, for longer than the timeout period. If the higher level lock cannot be acquired before the timeout value has been reached, then the process times out, all updates are rolled back, and all locks are released.



See Figure 2-5 for an illustration of the consequences of lock escalation for a partitioned table space. Partitions 1, 4, and 7 contain low level locks acquired by USER 1. Lock escalation is then triggered by USER 1. Note that escalation is tracked at the tablespace level, and not the partition level, so if the total number of locks on the entire tablespace exceeds the LOCKMAX value, lock escalation is triggered for all partitions that have been accessed by User 1. After the lock escalation, USER 1 now holds gross S locks on partitions 1, 4, and 7. The low level locks previously held by USER 1 are released.

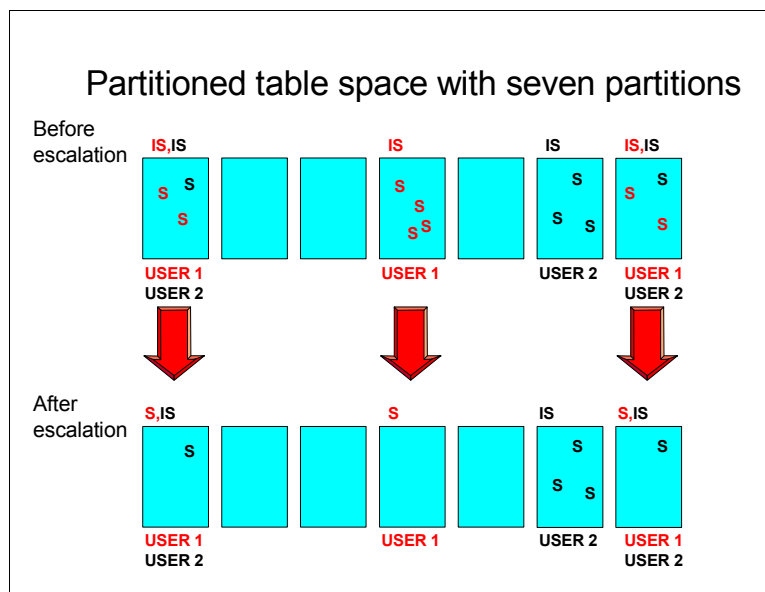


Figure 2-5 Lock escalation for a partitioned table space

See 5.19.1, “Suspension, deadlock, timeout, and lock escalation” on page 150 for more details about the lock escalation process.

## 2.2.4 Lock suspension and timeout

A process is suspended when it requests a lock on an object that is already locked by another concurrent process and the two locks are not compatible. The suspended process temporarily stops running. Incoming lock requests are queued in the order of arrival. Requests for lock promotion and requests for a lock by a process that already holds a lock on the same object take precedence over requests for locks by new processes. The suspended process resumes running when:

- All concurrent processes that hold incompatible locks release them.
- The requesting process times out or deadlocks and the process resumes to deal with an error condition.

The length of time that a suspension goes on for depends on the setting for a number of DSNZPARMs; these settings are documented in 7.3.1, “IRLMRWT: RESOURCE TIMEOUT field” on page 204.

For example: Using an Application for Inventory Control, two users attempt to reduce the quantity on hand of the same item at the same time. The two lock requests are queued. The second request in the queue is suspended and must wait until the first request releases its lock. See Figure 2-6 for a simple illustration of the lock suspension and timeout process.

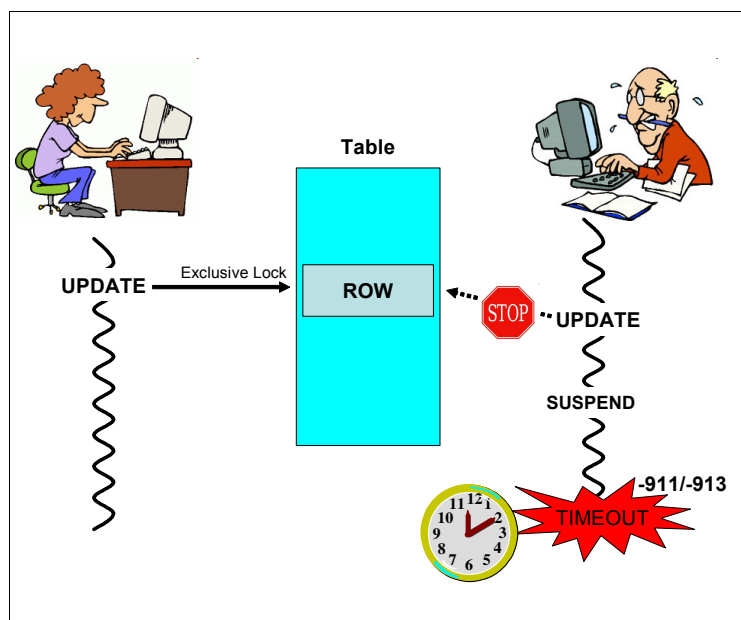


Figure 2-6 Lock suspension and timeout

## 2.2.5 Deadlock

A deadlock occurs when two or more processes hold locks on one or more resources that the others need and without which they cannot proceed.

IRLM, on behalf of DB2, scans all locks held for deadlocked processes at regular intervals as specified by the deadlock interval time DSNZPARM. See 7.3.2, "DEADLOK (part 1): DEADLOCK TIME" on page 207 for a description of DSNZPARM. IRLM notifies DB2 of all deadlocks and DB2 then either rolls back the current unit of work for one of the processes or requests a process to terminate.

To minimize the time taken to roll back, DB2 will choose the process to terminate, the *victim*, based on whether updates have been made to a NOT LOGGED table space and based on the number of log records written. If one process has updated a NOT LOGGED table space but the other has not, then DB2 will choose the victim as the one with no NOT LOGGED table spaces involved. This is because if the process that updated the NOT LOGGED table space was chosen then the table space would be placed in a recovery pending state. This happens regardless of how many log records have been written by the two processes. If both processes have updated NOT LOGGED table spaces, then the one that has written the least log records is the victim. The rollback will free the locks held by the victim and allow the remaining process to continue.

See Figure 2-7 for a simple illustration of a deadlock. Suppose:

1. User 1 updates table M, and acquires an exclusive lock for page A, which contains record 000100.
2. User 2 updates table N, and acquires an exclusive lock for page B, which contains record 000300.
3. User 1 then attempts to SELECT from page B of table N, while still holding the lock on page A of table M. User 1's task is suspended, because user 2 is holding an exclusive lock on page B.
4. User 2 then attempts to SELECT from page A of table M while still holding the lock on page B of table N. User 2's task is suspended, because user 1 is holding an exclusive lock on page A.
5. IRLM detects the deadlock situation, one thread gets "killed" and the other one continues.

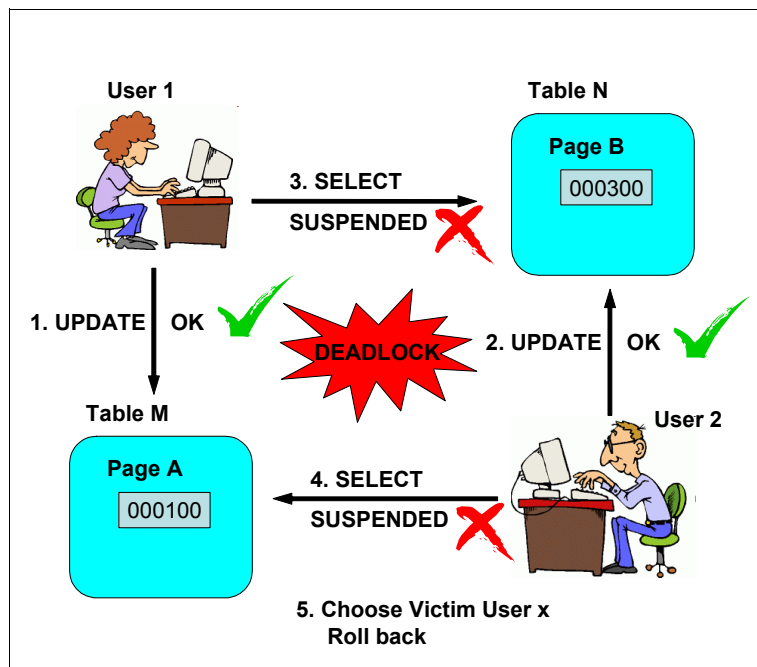


Figure 2-7 Simple illustration of a deadlock

There is no concept of an overall deadlock manager for application processes that access resources in multiple repositories, such as separate DB2 subsystems, or other resource managers, such as IMS or CICS. For example, it is possible for two processes to be running on separate DB2 subsystems, each trying to access a resource at the other location using a distributed access protocol, such as DRDA. In that case, neither subsystem can detect that the two processes are in deadlock; the situation is resolved only when one process times out.

## 2.3 Lock avoidance

As discussed previously, locking carries a cost both for concurrency and CPU. Provided certain conditions are met, DB2 is able to avoid requesting a Read or Share lock on behalf of the application process. This function, which only applies to low level locks, is referred to as *lock avoidance*. The application process has no direct control over when lock avoidance occurs; however, an understanding of the prerequisites can help an application designer make the best use of this valuable function.

One of the reasons for locking is to be certain that uncommitted data is not being read so that a read process normally requests an S lock. The S lock request would then queue behind any outstanding X lock for the same resource. In order for DB2 to read the row without taking a lock, DB2 must be sure that the data is committed. In the right circumstances, DB2 is able to ensure the “committedness” of the data without taking a lock.

DB2 uses the same lock avoidance techniques for index-only access. This has the added advantage that even if the row has been updated, lock avoidance can still take place provided that all of the data requested is in the index and that none of the index columns have been updated. This means that a subset of columns that are less frequently updated can be made available for read operations by putting them in an index. This does have the consequence that concurrency is access path dependent.

We call those rows in a DB2 page that satisfy the SQL predicate conditions and are returned to the program issuing the SQL call *qualifying rows*. Nonqualifying rows are also handled by DB2, but they are discarded and not returned to the program issuing the SQL call, as they do not satisfy the SQL predicate conditions.

In order to properly understand lock avoidance, it is first necessary to understand the various types of cursor that DB2 uses and the meaning of an “ambiguous” cursor.

### When lock avoidance occurs

Users have no direct control over the use of lock avoidance. Lock avoidance occurs in the following situations:

- ▶ Read-only or ambiguous cursor with ISOLATION(CS) and CURRENTDATA(NO).
- ▶ For any nonqualifying rows accessed by queries bound with ISOLATION CS or RS.
- ▶ When DB2 system managed referential integrity (RI) checks for dependent rows when either the parent key is updated or if the parent key is deleted and the DELETE RESTRICT option is defined.

Lock avoidance is more likely to occur when programs take frequent commits so that the duration of a unit of work is measured in low seconds rather minutes or even hours. See 8.3.4, “Periodic monitoring of DB2 system level concurrency related IFCIDs” on page 243 for guidance on monitoring of long running units of work. For CURRENTDATA(NO), lock avoidance is attempted for all rows (qualifying and nonqualifying). For CURRENTDATA(YES), lock avoidance can occur for nonqualifying rows only.

DB2 differentiates between three types of cursors:

- ▶ Read-only cursors
- ▶ updateable cursors
- ▶ Ambiguous cursors

A cursor is considered *ambiguous* if DB2 cannot tell whether it is used for update or read-only purposes. A cursor is ambiguous when any of the following statements are true:

- ▶ It is not defined with the clauses:
  - FOR FETCH ONLY or FOR READ ONLY. If these clauses are present, it is a *read-only cursor*.
  - FOR UPDATE OF. This makes the cursor *updateable*.
- ▶ It is not defined on a read-only result table (which would make it a *read-only cursor*).
- ▶ It is not the target of a WHERE CURRENT clause on an SQL UPDATE or DELETE statement. (This would make the cursor *updateable*.)

- ▶ It is in a plan or package that contains the SQL statements PREPARE or EXECUTE IMMEDIATE.
  - The presence of any dynamic SQL causes an ambiguous cursor, because DB2 cannot detect what follows in the program.
  - A cursor can appear to be read-only, but a dynamic SQL statement could modify data through the cursor. Therefore, the cursor is ambiguous.

Read-only cursors: If the result table is read-only, the cursor is read-only. The cursor that references a view with instead of triggers is read-only because positioned UPDATE and positioned DELETE statements are not allowed using those cursors.

The result table is *read-only* if one or more of the following statements is true about the SELECT statement of the cursor:

- ▶ The first FROM clause identifies or contains any of the following:
  - More than one table or view.
  - A catalog table with no updateable columns.
  - A read-only view.
  - A nested table expression.
  - A table function.
  - A system-maintained materialized query table.
  - A common table expression.
- ▶ The first SELECT clause specifies the keyword DISTINCT, contains an aggregate function, or uses both.
- ▶ It contains an SQL data change statement.
- ▶ The outer subselect contains a GROUP BY clause, a HAVING clause, or both clauses.
- ▶ It contains a subquery such that the base object of the outer subselect, and of the subquery, is the same table.
- ▶ Any of the following operators or clauses are specified:
  - The outer fullselect is not a subselect (contains a set operator).
  - A UNION, INTERSECT or EXCEPT operator.
  - An ORDER BY clause (except when the cursor is declared as SENSITIVE STATIC scrollable).
  - A FOR READ ONLY clause.
- ▶ It is executed with isolation level UR and a FOR UPDATE clause is not specified.

If the result table is not read-only, the cursor can be used to update or delete the underlying rows of the result table.

Table 2-3 shows the lock avoidance factors. “Returned data” in the third column means data that satisfies the predicates. “Rejected data” in the fourth column means data that does not satisfy the predicates. For ISO(RS), DB2 can only avoid locks on rejected data if the predicate is stage 1 or using multi-row fetch.

*Table 2-3 Lock avoidance and cursor access*

Isolation	CURRENTDATA	Cursor type	Avoid locks on returned data	Avoid locks on rejected data
UR	N/A	Read-only	N/A	N/A
CS	YES	Read-only	NO	YES
		updateable		
		Ambiguous		
	NO	Read-only	YES	
		updateable	NO	
		Ambiguous	YES	
RS	N/A	Read-only	NO	YES
		updateable		
		Ambiguous		
RR	N/A	Read-only	NO	NO
		updateable		
		Ambiguous		

### 2.3.1 Lock avoidance control

DB2 uses a combination of mechanisms to control lock avoidance:

- ▶ Page latching controlled by DB2 is a very efficient way to ensure physical consistency of the page, just as locks ensure that data is committed.
- ▶ The page log relative byte address (RBA) or log record sequence number (LRSN) in data sharing in the header of every page in the page set. The page log relative byte address value represents the point in the log when the last change was made to that page.

Plans and packages have a better chance for lock avoidance if the bind options used are ISOLATION(CS) and CURRENTDATA(NO).

For nonqualifying rows, lock avoidance can be applied for all cursors. That is, if SELECT... FROM... WHERE... FOR UPDATE OF... is specified and not many rows meet the conditions in the WHERE clause, then lots of lock avoidance can take place.

DB2 uses also lock avoidance mechanisms when accessing the parent referential integrity structures. For update primary key SQL statements or delete SQL statements with a restrict referential integrity constraint, DB2 uses lock-avoidance techniques to search for dependent rows.

For unambiguous read-only cursors with CURRENTDATA(NO) specified, DB2 uses the data page lock-avoidance technique to read all rows in the table without taking any locks.

If you use read-only cursors with CURRENTDATA(NO), the stability of the qualifying rows is not protected by the lock. As soon as the row qualifies under the protection of a data page latch, the row is passed to the application, and the latch is released. Therefore, the content of the qualified row might have changed immediately after it was passed to the application. To continue processing further rows in a page, DB2 must latch the page again.

The application developer has no direct control over DB2's use of lock avoidance techniques. However, the developer can write programs that meet the needed criteria to qualify for lock avoidance, if doing so fits in with the application requirements.

DB2 catalog and directory access does not use lock avoidance techniques.

Lock avoidance can be monitored with DB2 trace facilities using IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS (abbreviated to OMEGAMON® PE in this book) reports. See 10.12, "Lock avoidance in data sharing" on page 362.

## **BIND options**

The following options can be specified at a plan or package level, both in bind and rebind processes:

- ▶ **CURRENTDATA:** With CURRENTDATA(NO), DB2 considers lock avoidance techniques to access the data. This is the recommended option and is the default in DB2 9. Lock avoidance is not considered for qualifying rows if the application is bound with CURRENTDATA(YES).

Default value: NO (as it was with DB2 V8)

- ▶ **ISOLATION:** Cursor Stability (CS) increases the concurrency and also the possibility of lock avoidance, and is therefore recommended.

Default value: CS (it was RR with DB2 V8)

ISOLATION can also be specified in the SQL statement using the WITH clause.

An example is the following SQL:

```
SELECT NAME, ADDRESS
FROM USER.TABLE
WHERE DEPT = 110
WITH CS;
```

The ISOLATION value specified using the WITH clause in the SQL statement overrides the value specified for the ISOLATION option for the plan or package.

## Lock avoidance, parallelism, and block fetch enablement

Table 2-4 summarizes the impact of the CURRENTDATA option for parallelism and block fetch for distributed applications.

Table 2-4 Impact of CURRENTDATA option

Is currency of data required?	Ambiguous cursor	Read-only cursor
YES	<ul style="list-style-type: none"><li>▶ Lock avoidance is not considered for ISOLATION(CS) applications.</li><li>▶ I/O and CP parallelism are not allowed.</li><li>▶ Block fetching does not apply for distributed applications.</li></ul>	<ul style="list-style-type: none"><li>▶ Lock avoidance is not considered for ISOLATION(CS) applications.</li><li>▶ I/O and CP parallelism are allowed.</li><li>▶ Block fetching applies for distributed applications.</li></ul>
NO	<ul style="list-style-type: none"><li>▶ Lock avoidance is considered for ISOLATION(CS) applications.</li><li>▶ I/O and CP parallelism are allowed.</li><li>▶ Block fetching applies for distributed applications.</li></ul>	

Note that in both cases of CURRENTDATA option, lock avoidance is considered for unqualified rows for ISOLATION(CS) applications.

## Lock avoidance and singleton select

A singleton select uses the INTO clause and can return zero or one row at most. For example:

```
EXEC SQL SELECT * INTO :EMPREC
FROM DSN8910.EMP
WHERE WORKDEPT = 'D11'
FETCH FIRST 1 ROW ONLY
END-EXEC.
```

For isolation cursor stability, DB2 does not request an S row or page lock on the qualified row irrespective of the CURRENTDATA setting. The cursor is closed immediately after the singleton SELECT is processed. If you need to take and keep a lock, then use the WITH RS or WITH RR clause, both of which can be further modified with the USE AND KEEP EXCLUSIVE/UPDATE/SHARE LOCKS clause.

## Lock avoidance and overflow rows

When you update a variable length row on a page, it may be that after the update, the row no longer fits in its original page. In that case, the updated row is stored on a different page. In order to avoid updating all the indexes that point to that row, a pointer record is created on the original page. Remember that an index entry contains a row identifier (RID), which consists of the page number and, indirectly, the offset within the page where the row is located. The pointer record points to the location of where the actual row can be found, which could be anywhere in the table space. If a later update of the row decreases its size, it may be put back into its original (home) page without any updates to the index.

You can have variable length rows when:

- ▶ Using variable length fields like VARCHAR and VARGRAPHIC
- ▶ Using DB2 data compression (COMPRESS YES)
- ▶ Altering a table to add a column, but you have not run the REORG utility to materialize all the rows to have the new column



- Using the ALTER statement to make a schema change, which increases the size of a column, but you have not run the REORG utility to materialize all the rows to the latest format

The good thing about using the overflow pointer is that we avoid updating the index(es). The disadvantage is that we potentially double the number of data I/Os and getpage requests. As there is no lock avoidance on the row or page containing the pointer record, irrespective of the CURRENTDATA setting, there will be about the same number of locks; DB2 does not request a lock for the overflow row.

You can check whether or not overflow records exist by looking at the FARINDREF and NEARINDREF information in SYSIBM.SYSTABLEPART. This information is updated by running RUNSTATS and provides information about the number of rows that have been relocated far (> 16 pages) or near(< 16 pages) their “home” page. We recommend running a REORG when (FARINDREF + NEARINDREF) exceeds 5% in data sharing and 10% when not data sharing.

To reduce the amount of overflow rows, and hence the number of double I/Os, getpages, and lock/unlock requests, you can use a higher PCTFREE value (FREEPAGE will not help in this case) and REORG frequently.





## Serialization techniques

DB2 uses several other techniques besides the serialization provided by locks to control access to resources. In this chapter, we describe an additional lock that is used for index access by RR readers, and the Internal Resource Lock Manager (IRLM), the DB2 address space responsible for managing DB2 locks.

We describe how DB2 manages its locks on the catalog and directory during CREATE, ALTER, and DROP operations, and the restricted states, such as STOP or CHKP, that can be set on DB2 objects by a DB2 utility or a DB2 command.

We also discuss other DB2 serialization techniques that are complementary to locking.

DB2 uses claims and drains to serialize between application processing and utilities and latches to serialize access to in-memory objects, such as pages in the buffer pool or the log write output buffer.

The following topics are covered:

- ▶ Index specific lock
- ▶ Mass delete lock
- ▶ The Internal Resource Lock Manager (IRLM)
- ▶ DB2 subsystem object locking
- ▶ Claims and drains
- ▶ Partition independence
- ▶ Restricted states
- ▶ Latches

## 3.1 Index specific lock

The end of file (EOF) lock is related to the provision of repeatable read (RR) semantics when accessing a table using an index. Locking for repeatable read transactions. The lock has a description INDEX END-OF-FILE LOCK in a lock trace.

Consider the following situation: Program A is bound with ISOLATION(RR), and is executing the following SQL statement:

```
SELECT COLA, COLB
FROM USERID.TABLE
WHERE COLC < 100;
```

For simplicity, assume row locking is used for the table space containing the USERID.TABLE. Consider that the USERID.TABLE contains the values 10, 70, 90, 110, and 130 for key COLC. Concurrently, Program B is assumed to be executing the following SQL statement:

```
INSERT INTO USERID.TABLE
(COLA,COLB,COLC)
VALUES('COLA','COLB',95);
```

Figure 3-1 illustrates the steps executed.

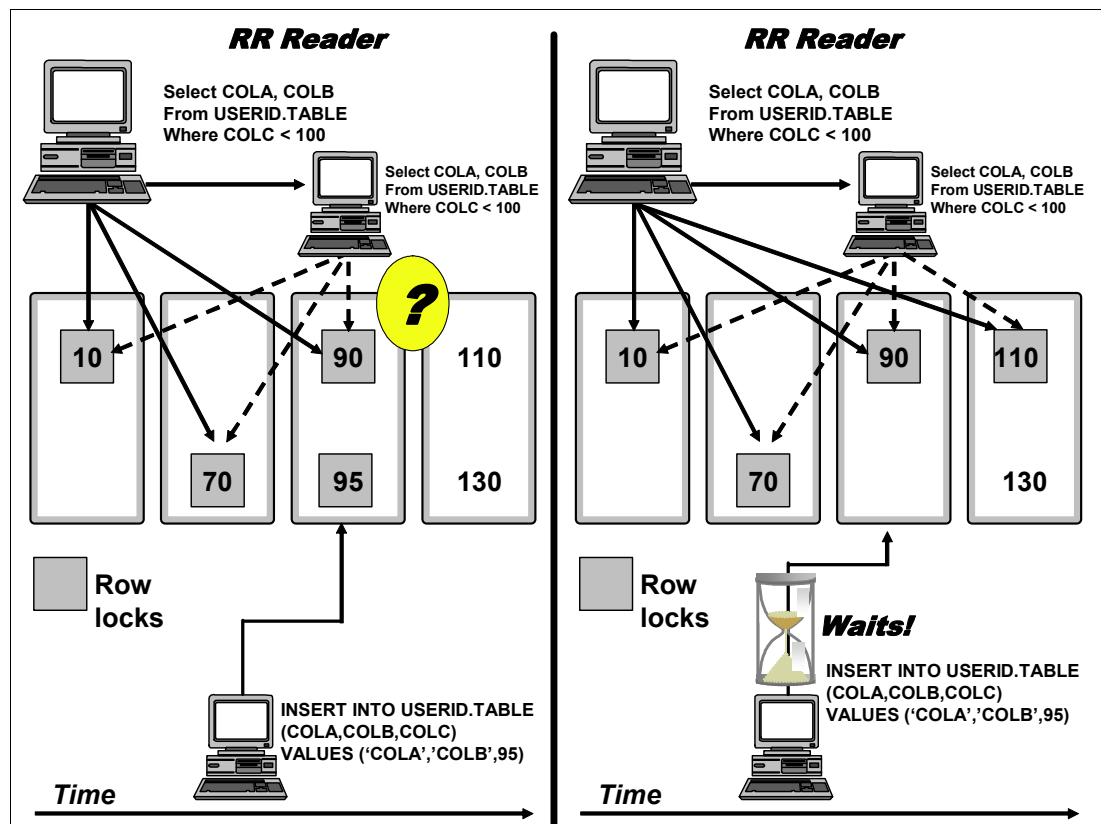


Figure 3-1 RR readers and concurrent insert process: semantics problem and implementation

If Program A is S-locking table rows with key COLC up to value 90, the Program B insert statement would execute and commit perfectly, but this would break the RR semantics for program A, as a subsequent read of the same table would provide an additional value for the newly inserted key 95.

To avoid these problems, RR readers take an additional S-lock in the first row that does not qualify as the RR predicate, that is, in this example, key 110. Note that this behavior does not occur if a table space scan is used for data access, as a high level share lock is used.

Inserting programs always check for the existence of an S-lock with an RR attribute in the next greater key than the one to be inserted; if an S-lock exists, the insert process is suspended until the RR reader commits and releases the lock, as insertion would violate the RR scanner's repeatability; if not, the insert is allowed.

Now consider the situation where the RR reader is reading to the last key of the index. The problem is the same. Assume Program A is:

```
SELECT COLA, COLB
FROM USERID.TABLE
WHERE COLC > 100;
```

There is no greater key to be locked to make sure the RR semantics are respected.

The EOF lock is used when an RR reader is executed and the highest key in the index satisfies the query. As in the previous example, the RR scanner needs to lock the first key that does not qualify (and no more keys exist), so the reader locks the end of index.

This means that until the RR reader commits, no high keys can be inserted in the index, as the EOF index lock prevents insertion of any key value greater than the highest existing key value. See Figure 3-2.

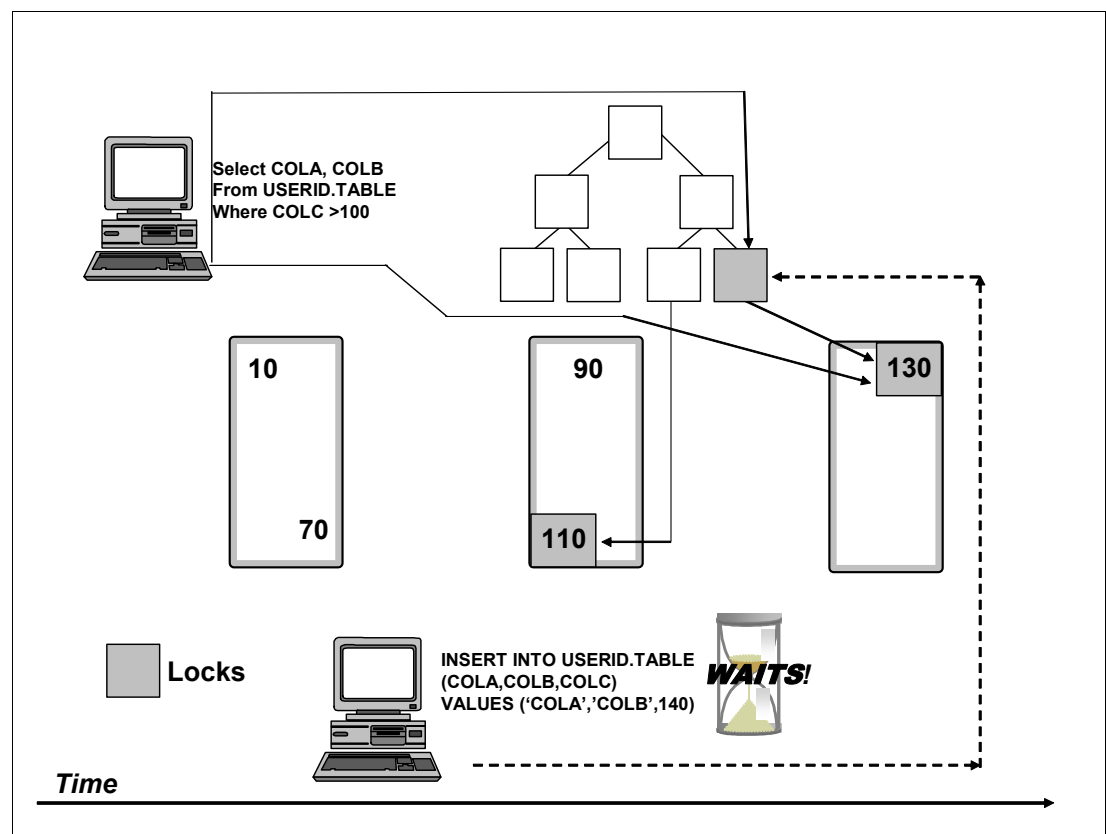


Figure 3-2 End-of-file index lock

## 3.2 Mass delete lock

Mass delete generally refers to a SQL DELETE statement without a WHERE clause, so all index entries and all rows are deleted. For example:

```
DELETE FROM USERID.TABLE
```

For simple table spaces and partitioned table spaces, performing mass delete requires that every data row be accessed and deleted.

For segmented table space and universal table spaces, both PBG and PBR, only the space map pages for the table need to be accessed so that the page segments can be marked “deallocated.”

The mass delete lock serializes a mass delete operation with an insert at the table level. The mass delete would get an exclusive mass delete lock. Then, insert (if a user wanted to allocate a deallocated segment of pages) would ask for a shared mass delete lock to make sure the mass delete that released the segment had committed, eliminating any possibility that the mass delete could roll back and reallocate the segment.

The mass delete lock is also used to serialize uncommitted readers on indexes with a mass delete.

## 3.3 The Internal Resource Lock Manager (IRLM)

DB2 application processes request data from DB2 by issuing a call to the DB2 language interface, which directs this request to DB2. DB2 then retrieves the requested data, but before the retrieval can happen, DB2 must check, by way of a call to the Internal Resource Lock Manager (IRLM), that this data is consistent and it is not being used by another concurrent application process. However, DB2 does not always have to use IRLM services. DB2 employs various lock avoidance techniques to try to establish the “committedness” of data without resorting to requesting a lock on the data. This is described in 2.3, “Lock avoidance” on page 33.

DB2 uses the IRLM to lock various resources in order to synchronize access to those resources. Locking can be explicit and implicit to the user. Explicit locking allows the users to lock specific table spaces, partitions, or tables by way of commands or SQL. Implicit locking of Database Descriptors (DBD), table spaces, partitions, or tables is generally performed during the resource allocation process. IRLM also locks pages or rows in table spaces. The locks prohibit more than one application process from updating the same data at the same time. For data sharing, DB2 ensures that all members of the data sharing group see the current version of a changed page by forcing the pages out at a commit or as a result of p-lock negotiations.

### 3.3.1 IRLM principles of operation

IRLM follows a number of strict rules when processing lock requests:

- ▶ The most important of these is that all requests by a process for a lock on a new resource are responded to on a first come first served basis. Essentially, all requests are therefore treated with the same priority so IRLM can be considered fair in its dealings with lock requests. This does have what could be considered undesirable consequences:
  - Process A holds an S lock on a resource, for example, a page.
  - Process B requests an X lock on the same page.
  - Process B is made to wait until process A releases the S lock, at which point IRLM will grant the X lock to process B.
  - Process C requests an S lock before process A releases its S lock. So at this point, the lock being held by process A is compatible with the lock being requested, but IRLM processes the requests in strict arrival sequence, and so process C must wait for both process A and process B to release the lock.
- ▶ Requests for lock promotion, and requests for a lock by an application process that already holds a lock on the same object, precede requests for locks by new applications. Within those groups, the request order is “first in, first out.”
- ▶ A process cannot have more than one lock on the same resource.
- ▶ If a process requests a more restrictive lock on a resource that it already owns a less restrictive lock on, then the less restrictive lock is promoted or upgraded to the more restrictive lock. For example, suppose a process has an S lock on a page. When a row on that page is updated by that process, then DB2 requests an X lock on the page. IRLM will promote the S lock to an X lock, assuming that there are other no other concurrent processes holding an incompatible lock on the page.
- ▶ A lock at a lower level of the hierarchy cannot be more restrictive than a lock held at a higher level by the same process. For example an IS lock at the table space level must be promoted to an IX lock before an S lock on a page in the table space can be promoted to an X lock.

### 3.3.2 Types of lock request

When DB2 requests a lock, the request can be qualified in a number of ways, as discussed in the following sections.

#### Conditional versus unconditional

An *unconditional* request effectively says “May I have a lock on this resource, and if I cannot, then let me know so I can suspend the active process until it becomes available”.

A *conditional* request effectively says “May I have this lock, but if it is not available, then just let me know so I can take another action”.

One example of the use of conditional locking is during insert processing. When inserting a row, DB2 has a preferred page on which to put the row, and as locking is DB2's mechanism of providing transaction semantics, DB2 is obliged to take an x commit lock on the page to perform the insert, assuming page level locking. DB2 makes this lock request a conditional one. If the lock on the preferred page is currently held by another agent, DB2 will find another candidate page for the INSERT rather than wait for the lock on the page of first choice.

If performing an update or delete of a row, however, locks are made unconditionally on the target page. There are no two ways about it: If you want to modify a row that lives on page P, that is the page that has to be locked.

## Lock duration

- Deallocation duration

The lock, if granted, is held until thread deallocation.

This type of lock request is used for high level locks with `RELEASE(DEALLOCATE)` bind parameter.

- Commit duration mode

The lock, if granted, is held until `COMMIT`.

This type of lock request is used for low level locks for `INSERT`, `UPDATE`, `MERGE`, and `DELETE` processing. Transaction semantics dictate that these changes are not shown until commit, so the locks that are used to enforce this task are taken with commit duration. Commit duration locks also apply to high level locks acquired on behalf of plans and packages bound with `RELEASE(COMMIT)`. DB2 also acquires locks for `COMMIT+1` duration for held cursors or held LOB locators.

- Cursor close duration

The lock, if granted, is held until all cursors are closed.

- Manual duration mode

The lock, if granted, is held until DB2 requests that it be released. This would normally occur before `COMMIT` and so shortens the duration of the lock.

This type of lock request is typically used for locks that provide data stability semantics. Thus, locks taken to provide `ISOLATION(RR)`, `(RS)`, or `(CS)` stability requirements are taken with manual duration and are released at a point when they are no longer needed for semantic enforcement. For example, consider an `ISOLATION(CS)` updateable cursor. The stability of what is under the cursor is provided until the cursor moves off the row. Accordingly, DB2 takes a manual duration lock on the row, and releases it when the cursor moves off the row (for example, when a fetch next through the cursor is performed). Conversely, an `ISOLATION(RR)` cursor receives repeatable results (modulo self) until commit. In this case, DB2 acquires manual duration locks, but does not release them until commit.

- Autorelease mode

The lock, if granted, is immediately released. This request is used when DB2 just needs to check whether a lock is already held by any other application process. IRLM tells DB2 whether or not the lock was available.

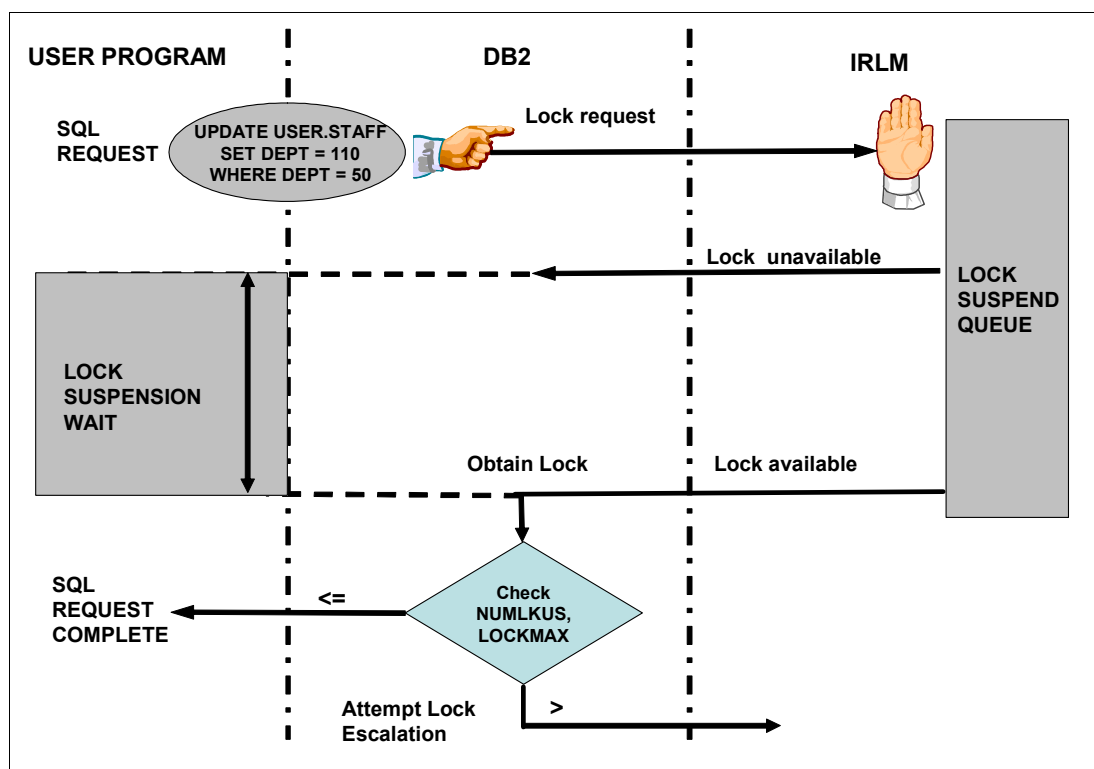
This type of lock request is used in just a couple of instances. One is in DB2 locking scenarios. This processing has to do with providing `ISOLATION(RR)` semantics during an index scan. Consider an `RR` query looking for rows with keys between 100 and 200, and consider the index containing keys 50, 150, 250, and 350. The query will lock qualifying rows (those rows with index keys 150, in this example), plus the row corresponding to the next higher key, 250. The locks will bear an `RR` attribute.



Now, an inserter trying to insert into this table while the RR query is running is free to insert, so long as nothing goes into the range of interest to the concurrently running ISOLATION(RR) process. So when performing an insert and there are RR queries in progress, DB2 will request an autorelease lock on the key next higher than the key DB2 is trying to insert. Feedback on the lock request tells if there were holders of the lock or not, and if there were holders, if there was the RR attribute on the lock. The lock itself is uninteresting. The feedback is what guides processing. If there is an RR attribute on the lock, the insert is denied (see 3.1, “Index specific lock” on page 42).

### 3.3.3 Lock processing

Figure 3-3 shows the sequence of steps when executing an SQL statement that requires a lock.



*Figure 3-3 Lock processing*

An SQL request (SELECT, FETCH, INSERT, DELETE, MERGE, or UPDATE) from an application thread results in a lock request from DB2 to the IRLM, unless lock avoidance is possible or the SELECT is using isolation uncommitted read (UR).

IRLM checks whether the requested resource is available for the type of lock requested. If the request can be granted, a new lock element is added to the lock manager's queue of elements and chained off the queue for the thread holding the lock.

If the request cannot be granted, it is queued on a suspend queue to wait until:

- The lock on the resource is freed by the holder reaching the end of a UR, or the holder moves to a different page through a cursor select and no update, or there is an explicit or implicit rollback as a result of abend by holder.
- IRLM determines that the suspended thread has timed out.

- IRLM determines that the suspended thread is a participant in a deadlock situation.

IRLM monitors the length of all suspensions on a timed interval, that is, the deadlock detection cycle time. When a deadlock exists or the specified installation values for the timeout period are reached, then deadlock or timeout processing occurs respectively. A deadlocked program receives a -911 or -913 SQLCODE while a program that times out receives a -911, -913, or -923 SQLCODE. A -911 SQLCODE indicates that the work will be rolled back, while for a -913 SQLCODE, the ROLLBACK versus COMMIT decision is passed back to the calling application process.

Table 3-1 shows the SQL return code in case of deadlocks in different environments.

Table 3-1 Deadlock SQL return codes

Result of Deadlock	Environment						
	TSO and CAF	TSO and CAF	IMS MPP	IMS BMP	CICS RCT ROLBE =NO	CICS RCT ROLBE =YES	Stored Procedure <sup>a</sup> , UDF or DBAT
<b>BACKOUT</b>	Yes	No	Yes	Yes	No	Yes	No
<b>PSEUDO ABEND</b>	—	—	U777	—	—	—	—
<b>SQL code</b>	-911	-913 <sup>b</sup>	-	-911	-913 <sup>c</sup>	-911	-913

a. If the timeout occurs during open processing, then a -913 occurs; if the timeout occurs during FETCH from a result set by the calling application, then the same code occurs as for a calling application.

b. Only in a DB2 abend situation.

c. Only the current SQL statement is backed out.

After granting a requested lock, DB2 checks that:

- The total number of page or row locks currently held for the thread does not exceed the specified installation maximum (NUMLKUS).
- The total number of page or row locks currently held for the thread on the target table space does not exceed the specified thresholds for the table space (installation maximum NUMLKTS and table space create maximum LOCKMAX for lock escalation to occur).

Exceeding NUMLKUS results in the return of a unique SQLCODE -904 to the calling program, which can be interpreted as indicating the need for more frequent COMMIT points in your program.

Exceeding LOCKMAX will trigger lock escalation in case the table space being accessed is defined with a locksize of ANY, PAGE, or ROW.

Therefore, the application program can receive two possible results from a lock request: The program either gets the requested lock and continues processing with or without a lock suspension wait or receives an SQL return code indicating an abnormal situation was detected, such as exceeding NUMLKUS or a timeout or deadlock.

Once DB2 has finished processing the data, and depending on the isolation options, a call to the IRLM is performed to release the lock on the object.

DB2 is also responsible for coordinating logging and commit functions. This requires a thread wide release of all locks held by a thread that is terminating or doing commit processing. While releasing the locks, IRLM also tests if there is another thread on the suspend queue waiting for the object being unlocked. If so, that thread can be granted the lock and end its lock suspension wait.

The lock management requires pure CPU time. Except for possible paging, there is no input or output operation or any inherent delay in the lock management other than lock contention.

The CPU cost for lock management work is a function of:

- ▶ The frequency of locking requests issued by DB2
- ▶ The overall number of existing locks
- ▶ The frequency with which threads terminate or commit
- ▶ The frequency with which the IRLM deadlock cycle is initiated
- ▶ The number of suspended locks

### 3.3.4 IRLM service class definition in Workload Manager (WLM)

The IRLM address space always needs to be assigned to a service class whose goal would give it one of the highest MVS dispatching priorities; usually SYSSTC will suffice. This is because of the importance of detecting and breaking deadlocks, for error management, and for lock resumption. IRLM needs this priority because it manages resources that are managed by work within the DB2 address spaces.

Figure 3-4 shows the types of address spaces that are typically active when DB2 is running and the recommended relationship between the service class goals.

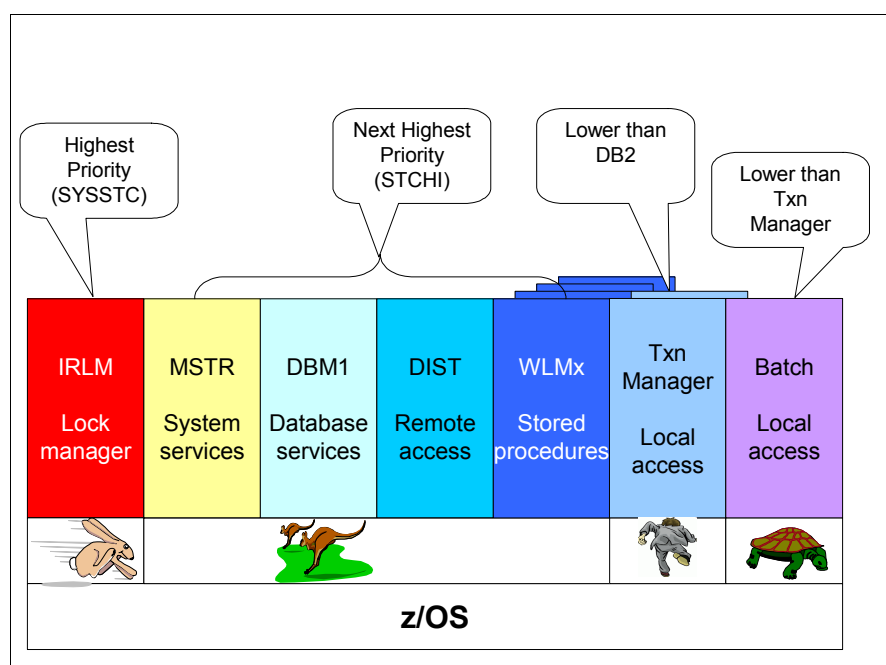


Figure 3-4 IRLM service class definition in Workload Manager

Use the following service classes for non-DBMS address spaces:

- ▶ SYSSTC service class for:
  - VTAM® and TCP/IP address spaces
  - IRLM address space (IRLMPROC)

**Important:** The VTAM, TCP/IP, and IRLM address spaces must always have a higher dispatching priority than all of the DBMS address spaces, their attached address spaces, and their subordinate address spaces. Do not allow WLM to reduce the priority of VTAM, TCP/IP, or IRLM to the same or below the priority of the other database management system (DBMS) address spaces.

- ▶ An installation-defined service class with a high velocity goal for:
  - DB2 (all address spaces, except for the DB2-established stored procedures address space):
    - ssnmMSTR
    - ssnmDBM1
    - ssnmDIST (DDF address space)

When you set response time goals for distributed threads or for stored procedures in a WLM-established address space, the only work that is controlled by the DDF or stored procedures velocity goals is the DB2 service tasks (work performed for DB2 that cannot be attributed to a single user). The main reason for including DIST in this list is that it is considered a service address space to MVS and thus it needs to have sufficient priority to get new work started into the address space. Once the work has been started, it will then be run under an independent WLM enclave that will be given a priority commensurate with the kind of work that it is performing. Do not make the mistake of having the DIST address space separate in priority from the other main DB2 address spaces. The user work runs under separate goals for the enclave.

For the DB2-established stored procedures address space, use a velocity goal that reflects the requirements of the stored procedures in comparison to other application work. It should be lower than the goal for DB2 address spaces. These address spaces are MVS service address spaces and need the priority to get work done without interruptions.

Consider the following other workload management considerations:

- ▶ IRLM must be eligible for the SYSSTC service class. To make IRLM eligible for SYSSTC, you do not need to classify IRLM to one of your own service classes.
- ▶ If you need to change a goal, change the velocity between 15 and 20%. Velocity goals do not translate directly to priority. Higher velocity tends to have higher priority, but this is not always the case.
- ▶ If you use I/O Priority management, WLM can assign I/O priority (based on I/O delays) separately from processor priority.
- ▶ z/OS workload management dynamically manages storage isolation to meet the goals you set.

The DB2 address spaces should be set at a higher service class with a higher goal than the transaction managers.

### 3.3.5 DB2 and Workload Manager application process prioritization

There are times when a high priority application defined to WLM needs to access or update a row that is exclusively locked by a low priority process, such as a batch job. The high priority process must wait until the low priority process issues a commit.

Assume that a transaction executes with a low WLM priority and makes updates to the database. Because it is updating, it must acquire X locks on the pages that it modifies and then these X locks are held until the transaction reaches a commit point. This is business as usual. If such a transaction holds an X lock on a row or page that another transaction is interested in, the other transaction is forced to wait until the first transaction (with a low WLM priority) commits. If the lock waiter performs important work and is thus assigned to a high WLM service class, it would be preferable that it is not forced to wait by other transactions that execute in a low priority service class.

To reduce the time the high priority transaction has to wait for that lock to be given up by the low priority transaction, it would be better if the waiting transaction could temporarily assign its own priority to the transaction that holds the lock until this transaction frees the locked resource. The WLM component of z/OS provides a set of APIs that can be used to accomplish this task, which is referred to as *WLM enqueue management*.

## 3.4 DB2 subsystem object locking

There is some locking activity that is not related to user data, but takes place in shared DB2 subsystem objects. This locking activity can be caused by user application programs. It can also be caused by DB2 system plans processing.

You are never aware of most of this locking activity, although in some cases it can produce suspension problems in your system, such as a timeout DSNT376I message listing the plans involved.

The DB2 system plans that generate locking activity are:

- ▶ BCT, which is the plan used to perform service tasks and handle requests from DB2 resource managers. A lock issued by this plan is usually of short duration.
- ▶ ACT, which is the authorization plan used in the process of validating a user's authority to access a specified plan. A lock issued by this plan is of short duration, but ACT is used frequently, with every create thread.
- ▶ DSNBIND, which is the plan used for all binds. It is major contributor to the length of time it takes to complete a bind.
- ▶ DSNUTIL, which is the plan used by DB2 utility control program DSNUTILB. The duration of locks held by this plan varies according to the specific DB2 utility function being invoked.

The DB2 subsystem objects involved in locking activity are:

- ▶ DB2 catalog and directory
- ▶ Skeleton cursor table (SKCT) and skeleton package table (SKPT)
- ▶ Database descriptors (DBDs)

### 3.4.1 Locks on the DB2 catalog and directory

The DB2 catalog, which consists of tables of data about everything that is defined to the DB2 system, and the DB2 directory, whose tables contain information that DB2 uses to control normal operation, are subject to update activity that must be serialized.

Although the catalog and directory are designed to minimize contention among application processes, update activity may lead to locking problems in concurrency situations.

Five main activities can cause suspension problems in the DB2 catalog and directory:

- ▶ DB2 tasks that update DB2 catalog and directory tables. An example is writing to SYSLGRNX every time a table space or partition is opened and updated.
- ▶ The BIND, REBIND, and FREE process, which reads some DB2 catalog tables, such as SYSIBM.SYSTABLES, and inserts or updates others, such as SYSIBM.SYSPACKAGE.
- ▶ Data definition processes, such as CREATE, ALTER, and DROP, which also insert, update, or delete catalog and directory entries (for example, DBD01).
- ▶ Data control processes, such as GRANT and REVOKE, which affect the concurrency of the catalog authorization tables (for example, SYSIBM.SYSUSERAUTH).
- ▶ Utility processing, which may also lock some directory tables (for example, SYSUTILX).

To avoid most of these concurrency problems, process all the data definition, data control, and BIND activities in a dedicated window outside of production hours.

See Chapter 6, “Utilities, commands, and SQL” on page 161 for details.

### 3.4.2 Locks on skeleton tables

The SKCT is located in the SCT02 table space in the directory and describes the structure of SQL statements in application plans. The SKPT is located in the SPT01 table space in the directory and applies to packages.

When you bind a plan, DB2 creates an SKCT in SCT02. When you bind a package, DB2 creates an SKPT in SPT01.

The following operations require exclusive control of the related SKCT and SKPT as they are updating them:

- ▶ Using BIND, REBIND, and FREE for the plan or package
- ▶ Dropping a resource or authority on which the plan or package depends
- ▶ In some cases, altering a resource or authority on which the plan or package depends

When you run a plan or package, DB2 takes a shared lock on it, so that changes cannot be made to the object while it is being executed in an application program. Thus, the execution of a plan or package contends with operations listed above.

### 3.4.3 Locks on database descriptors

The database descriptors (DBDs) describe the DB2 databases. Each DBD fully describes a database and all of its objects, such as table spaces, tables, indexes, and referential relationships, and contains other access information.

DBDs are located in the table space DBD01 in the directory.

Two main processes have to read and lock a DBD in shared mode:

- ▶ Dynamic SQL statements
- ▶ Active utilities

Static SQL does not take any locks on the DBD if the DBD is cached in the environment descriptor management (EDM) pool. Therefore, DB2 does not have to go to the DB2 directory table space to retrieve the DBD.

Each time a database or a dependent object definition is modified, the DBD object in the directory must be updated. This update process requires an exclusive lock on the DBD, which is incompatible with dynamic SQL and utility execution. If the DBD is in use for a plan or package, dynamic SQL and utility execution are suspended.

You might consider the granularity of your database definitions to improve concurrency.

Table 3-2 summarizes the DB2 subsystem locking activity.

Table 3-2 DB2 subsystem locking activity DBD - (d)

Object of Locking	DB2 process							
	Static SQL	Dynamic SQL	BIND	Create table	Alter table	Drop table space	Grant	Revoke
Catalog table spaces	IS <sup>a</sup>	IS <sup>b</sup>	IX	IX	IX	IX	IX	IX
SKCT or SKPT	S	S	X	—	X <sup>c</sup>	X <sup>d</sup>	—	X
DBD	— <sup>e</sup>	S	S	X	X	X	—	—

- a. IS locks on the catalog table spaces are held only for a short time to check EXECUTE authority if the plan or package is not public or the authorization list is not cached in the EDM pool.
- b. Except when checking EXECUTE authority (see Note a), IS locks on the catalog table spaces are held until the COMMIT point.
- c. SKCT or SKPT is marked invalid if a referential constraint (such as a new primary key or foreign key) is added or changed, or the AUDIT attribute is changed in the table.
- d. SKCT or SKPT is marked invalid as a result of a drop table space operation.
- e. If the DBD is not in the EDM pool, S-locks are acquired on the DBD table space, which effectively locks the DBD.

## 3.5 Claims and drains

In an online DB2 environment, the compatibility between transactions or batch programs and utilities or DB2 commands can be an issue. Some utilities and commands require full control over an object for some of the phases of their operation, which may be difficult to achieve, as there may be applications that have outstanding locks on the object. DB2 solves this problem by using claims and drains, which enable DB2 utilities, DDLs, commands, and SQL EXCHANGE statements to take over access to some objects independent of any transaction locks held on the objects. Claims and drains are a complementary serialization mechanism, in addition to locks, which are managed wholly by DB2.

## 3.5.1 Claims

DB2 uses a claim to register that a resource is being accessed. A process that stakes a claim on a resource is in effect telling DB2 that "Hey, I am using this!"

The following resources can be claimed:

- ▶ Simple table spaces
- ▶ Segmented table spaces
- ▶ A single data partition of a partitioned table space
- ▶ A non-partitioned index space
- ▶ A single index partition of a partitioned index (of either the partitioning index or a DPSI)
- ▶ Logical partitions of non-partitioned indexes
- ▶ Universal table space with CLONE

A claim is acquired at first access of a resource. Claims prevent drains from acquiring a resource. Claims and drains provide another "locking" mechanism to control concurrency for resources between SQL statements, utilities, and commands. As with transaction locks, claims and drains can time out while waiting for a resource. Each claim has a claim class associated with it that is assigned based on the type of access being requested, much like a lock type. Table 3-3 shows the three possible claim classes.

*Table 3-3 Claim classes*

Claim classes	Definition
CS	Read with ISOLATION(CS)
RR	Read with ISOLATION(RR)
WRITE	Delete, Insert or Update activity

DB2 monitors claims at the page set level (table space, index space, or partition) for each of these classes. Claim processing is not dependent on the ACQUIRE option of the BIND command or any other BIND, SQL, or installation option. Claims are acquired only for partitions accessed within the commit scope.

Claims are always released at a commit point unless there is a cursor defined as WITH HOLD.

## 3.5.2 Drains

A drain prevents new claims and waits for existing claims to be released. Drains are requested by DB2 commands and utilities and SQL EXCHANGE. A drain may only need partial control. It can drain a combination of claim classes:

- ▶ Only the write claim class
- ▶ Only the repeatable read claim class
- ▶ All claim classes

A drain is the action of taking over access to an object by preventing new claims and waiting for existing claims to be released.

For example, a utility can drain a partition when applications are accessing it. The drain quiesces the applications by allowing each one to reach a commit point, but preventing any of them, or any other applications, from making a new claim. When no more claims exist, the process that drains (the drainer) controls access to the drained object.



Drains can be requested by DB2 commands, utilities, and DDL.

For example, the CHECK INDEX utility needs to drain only writers from an index space and its associated table space. RECOVER, however, must drain all claim classes from its table space. The REORG utility can drain either writers (with DRAIN WRITERS) or all claim classes (with DRAIN ALL).

The drain acquires a special lock, which is known as a drain lock. It is a real IRLM lock and blocks new claimers. It then waits for all current claims on the object to be released.

The effect of the drain is to make active applications quiescent by allowing them to reach a commit point while preventing new applications from accessing the object by making a new claim.

The purpose of the drain lock is to make any claim processes wait for the drain process to complete. When the object is drained, a drain-all will get an X drain lock, but drain writers gets an IX drain lock on the object. The claimer then requests a shared (S) drain lock and waits for the drain lock to be released.

Another case is the EXCHANGE SQL statement process for CLONE tables. In order to prevent the objects that are being exchanged from having concurrent access with DML or utilities, the EXCHANGE process drains both base and clone objects.

### 3.5.3 Interaction between claims and drains

Claims have to request and wait for the drain lock to be released only when a drain is in control of the page set. Otherwise, the increment of the claim count by the claim occurs without acquiring the lock. When a claim acquires the lock, it holds it only until the count is incremented. The claim is granted and the claim count is incremented as long as no incompatible exception states are present.

When no more claims are present on the object (the claim count is set to zero), the DB2 utility or command takes control of the object.

There are three drains that correspond to the three claim classes:

- ▶ CS: Drains CS claims.
- ▶ RR: Drains RR claims.
- ▶ WRITE: Drains WRITE claims.

The drain lasts until the DB2 command or the utility finishes processing.

So far we have seen that utilities and commands do not follow the normal IRLM locking process to serialize with running transactions, and thus they do not claim objects. Noticeable exceptions to this statement are:

- ▶ COPY SHRLEVEL(CHANGE) and RUNSTATS SHRLEVEL(CHANGE) both request a read claim; they do not drain any class.
- ▶ During the UNLOAD phase, REORG SHRLEVEL CHANGE only claims.

#### Compatibility rules

The concurrent execution of two different utilities is not typically controlled by the drain and claim process; instead, utilities use a set of compatibility rules that dictate that a new utility can start only after it is checked against all other utilities running on the same object and found to be compatible.

Basically, some utilities, such as the LOAD utility, require exclusive read and write access to the object. Others, such as COPY SHRLEVEL(REFERENCE), allow concurrent read-only access.

For information about the compatibility of utilities, refer to each utility description in the *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.

## 3.6 Partition independence

Partition independence allows parallel access to different partitions of a partitioned table space and its related partitioned or logically partitioned indexes. This applies to the execution of utility jobs, SQL applications, and DB2 commands. Partition independence applies to table spaces that are partitioned, that is, universal partitioned-by-growth (PBG) and universal partitioned-by-range (PBR) table spaces.

We provide more information about this topic in 6.4, “Partition independence” on page 172.

## 3.7 Restricted states

Restricted states can be set on DB2 objects by a DB2 utility, a DB2 command, or DDL to control access and help ensure data integrity. For example, when the database administrator needs to repair a table space, the DBA is able to issue a STOP command to restrict access to a single partition, a whole table space, or an entire database.

We describe restricted states in 6.8, “Restricted states” on page 193.

## 3.8 Latches

DB2 uses latches to serialize access to resources such as pages in the virtual buffer pool and the log write output buffer. The most important of these, from a user perspective, is the virtual buffer pool page latch described in 3.8.1, “Buffer pool data and index page latching” on page 56. Latches are also used to serialize access to many other DB2 resources. These latches are described in 3.8.2, “DB2 latches” on page 58.

### 3.8.1 Buffer pool data and index page latching

DB2 uses two levels of serialization: locking and latching.

Locking is performed for DB2 by the Internal Resource Lock Manager (IRLM) to protect sections of a database. Because locking requires communication with another subsystem, it takes longer to perform than latching.

Latching is used for short-term serialization of DB2 resources, such as storage or control blocks. Latches on data are always at the page level. They were introduced to the product with row level locking. When DB2 only had page level locking, there was no need for latches. We explain why this is so after we talk about row locking and page latches.

Row (and page) locks are called “transaction locks”. They are used to provide transaction consistency. I lock the data to update it. You lock the data to update it. Until I commit, you cannot acquire the lock. You are thereby stopped from seeing my uncommitted update.

Further consider the micro operations that compose updating a data row upon a page. The row may shrink or grow in size. If it shrinks, the extra area becomes a "hole" (is chained into a chain of available space on the page). If it needs to grow, other rows on the page may have to be moved around so we can coalesce holes to make enough contiguous space on the page to accommodate the now larger row. In addition, we modify the count of free bytes on the page. It would not be good for a reader to look at and try to retrieve data from a page while these physical changes are taking place, even if the interest is in a row other than the one we are updating.

The latch is a low level serialization mechanism used to ensure physical consistency of shared DB2 data structures or resources (for example, pages in the buffer pool), implemented to be very inexpensive to acquire (short path length), and very basic in functionality (no deadlatch detection). It is used to provide the physical consistency of a page. Hence, when a row is being updated under row level locking, a row lock is held on the row and a page latch is held on the page. As soon as the modification is done (but not committed), the latch is released, as the page is consistent. Others may now look at the page and not discover physical inconsistencies. The row lock will be held until commit in order to provide transaction consistency.

A latch is a memory-based serialization technique (as opposed to a "lock", which is name-based). Latches are faster than locks, and are generally held for shorter durations. In some cases when page locks are not used (for example, row level locking or space map page updates), DB2 relies on the "page latch" to ensure that only one transaction at a time is physically modifying the page.

Page latches are held for a shorter duration only while a program is accessing a page, cannot last after a commit point (locks taken in a `CURSOR WITH HOLD` do last), and are requested in such a way that no timeouts or deadlocks involving latches can occur.

To prevent latch-lock deadlocks, a latch is never held by a process while waiting for an IRLM lock. However, a process can hold more than one latch at a time.

When an update to the data page requires information to be updated in the space map page, DB2 latches the data page before latching the space map page to avoid the possibility of a deadlatch.

There are only two types of page latches: S (shared) and X (exclusive).

- ▶ The S-latch is used to avoid acquiring IRLM locks. Readers use the S-latch to ensure the physical consistency of the page. If the commit state of the data page or row cannot be determined, an IRLM lock is requested, unless ISO(UR) semantics are used.
- ▶ The X-latch is used to serialize access with S-latches and other X-latches for the update process. Before a process can update the data on the page, it must acquire both an X-latch and an X-lock.

Requests for a page latch are queued in the same way that IRLM queues requests for a lock. The page latch suspension time is reported in the accounting class 3 suspension times under the heading `PAGE LATCH`. See "DB2 page latch suspension" on page 279 for a discussion about how to monitor and trace page latch contention.

Table 3-4 summarizes the differences between the latch and the lock.

*Table 3-4 Differences between the latch and the lock*

<b>Latch</b>	<b>Lock</b>
Ensures physical consistency and is inexpensive.	Ensures logical consistency but is relatively expensive.
Deadlatch is not possible.	Deadlock is detected.
Latch can be suspended and timeout detection is required.	Lock can be suspended and timeout detection is required.
The object of a latch is a page.	The object of a lock is a table space, partition, table, page, or row.
Duration is very short.	Duration is short to long.
Latch types are S and X.	Lock modes are IS, IX, SIX, S, U, and X.

### 3.8.2 DB2 latches

DB2 uses latches to serialize access to many different memory resources, such as the log output buffer, storage, or control block chains. A latch class hierarchy is used to prevent a “deadlatch” by making sure that latches are always obtained in the same sequence by all requestors.

DB2 counts the number of times a process has to wait for a latch in a number of different statistics latch class counters. Since APAR PK77514 (PTF UK43998), which removed the serviceability specification for latch class counters, the description of each class is documented in the DSNDQVLS macro and can be found in SDSNMACS. See Table 3-5 for the description of each of these classes.

*Table 3-5 Latch class*

<b>Latch class</b>	<b>Primary content</b>
1	Infrequently used
2	Global authorization cache
3	DDF disconnect
4	SYSSTRING cache
5	IRLM data sharing exits or RLF
6	Data sharing index split
7	Index latch and OBD allocation
8	Query parallelism
9	Utilities or stored procedure URIDs
10	Allied agent chain or sequence descriptors Create thread queued by reaching CTHREAD limit
11	DGTT allocation Sequence or identity column
12	Global transaction ID table

Latch class	Primary content
13	Pageset operations
14	Buffer pool hash chain and LRU chain
15	ARCHIVE LOG MODE(QUIESCE)
16	UR chain
17	RURE chain
18	DDF resynch list
19	Log write
20	System checkpoint
21	Accounting rollup
22	Internal checkpoint
23	Buffer manager: <ul style="list-style-type: none"> <li>► Add page latch waiter on timer queue</li> <li>► Add remove to/from deferred write queue for GBP-dependent objects</li> </ul>
24	EDM pool LRU chain Buffer Manager page unlatch and prefetch
25	Workfile allocation EDM hash chain
26	Dynamic statement cache
27	Stored procedures queue
28	Stored procedures or authorization cache
29	Field procs and DDF transaction manager
30	Agent services
31	Storage manager
32	Storage manager
254	Index latch

The first point is that there are many more types of latches than the number of latch classes, currently 33, and so each latch class is counting the number of times DB2 has to wait for a latch for multiple types of latches. In order to be sure which type of latch is actually causing the counter to be incremented, it is necessary to run a trace. See “DB2 latch suspension detailed information” on page 279 for a description of how to trace DB2 latch contention.

The next point is that DB2 does not accumulate these counters in the way that you might expect. If process A has latch L1 and process B is waiting for that latch, then one is added to the latch class for that latch type. This is as you might expect. However, suppose that this time process A has a latch L1 and process B and C are waiting for the same latch. This time two is added to the counter, one for each waiter. Now when process A releases the latch, process B and C race to get the latch as, unlike locks, there is no latch queue. The winner gets the latch, but the loser again has to wait, so one is added to the latch wait class counter. The counters can therefore become very large very quickly whenever latch contention starts to occur. There is therefore no correlation between these statistics counters and the number of IFCID56/57

trace records, because an agent may be suspended multiple times or not at all while trying to obtain a latch.

The latch class statistics counters are just that: counters. They are reported by OMEGAMON PE in the statistics long report as the number of times a process waits per second. If you need to know the latch duration, then take a look at the accounting report class 3 suspensions, where you can find the field LOCK/LATCH(DB2+IRLM).

## Latch classes of interest

The following latch classes are of most interest, though bear in mind that the latch classes all count latch contention for multiple types of latch wait. A simple rule of thumb for these latch contention counters is shown in Table 3-6.

Table 3-6 Latch class contention counts: when to pay attention

Counter value	Action
Less than 1,000 per second	Unlikely to be an issue
Between 1,000 and 10,000 per second	May be an issue; monitor for change
Above 10000 per second	Likely to be an issue and should be investigated

### LC06: index tree P-lock held during an index page split

This is latch type 70 (or X'46') in the performance trace. This only applies to data sharing. The equivalent for non-data sharing is counted in latch class 254, which is also latch type 254 (or X'FE'). The latch is held during index tree structure modification.

Remedies for high values are:

- ▶ Reduce index splits by use of the REORG utility to add free space.
- ▶ Minimize index key size, especially for a unique index.
- ▶ Use the NOT PADDED option on CREATE INDEX or ALTER INDEX for indexes with VARCHAR columns with a length of more than 18 bytes. If you have VARCHAR columns with a length more than 18 in indexes, consider altering these indexes to NOT PADDED, rebuild the indexes, and rebind dependent packages that may be able to exploit the index-only access.
- ▶ Consider using larger page sizes in DB2 9 to reduce the frequency of index splits.
- ▶ Note that DB2 9 introduced.

The number of index splits can be estimated from LEAFNEAR/FAR in SYSINDEXPART and also REORGLAUFNEAR/FAR in the real-time statistics table INDEXSPACESTATS.

### LC14: virtual buffer pool latch

This class is used to count the latch waits for managing the hash chains when PGSTEAL=LRU.

Remedies for high values are:

- ▶ Use FIFO if the objects are resident in memory so that no or minimal read IO occurs.
- ▶ Increasing the size of the pool can reduce hash contention. You must ensure that any increase is backed by real storage and does not result in z/OS paging to auxiliary.
- ▶ Assign the objects across more buffer pools to balance the workload as measured by the rate of GETPAGE requests.

### ***LC19: log write latch***

This latch class counts the number of times a wait occurs for the log buffer to become available.

Remedies for high values are:

- ▶ Minimize the number of log records created by:
  - LOAD RESUME/REPLACE with LOG NO instead of massive INSERT/UPDATE/DELETE.
  - Segmented or universal table space if mass delete occurs.
- ▶ Increase the size of the log output buffer if there is a non-zero unavailable count.
  - Unavailable is reported in field UNAVAILABLE OUTPUT LOG BUFF in the statistics report.
  - When available, the first agent waits for the log write; all subsequent agents' waits are counted in LC19.
- ▶ Reduce the size of the output log buffer if there is non-zero output log buffer paging. Paging is reported in field OUTPUT LOG BUFFER PAGED IN in the statistics report.
- ▶ Reduce use of class 3 accounting if there is more than 10,000 latch contentions per second.

### ***LC24: EDM pool LRU latch***

This is latch type 18 (or X'24').

Remedies for high values are:

- ▶ Use DSNZPARM EDMBFIT=NO.
- ▶ Enable thread reuse with RELEASE DEALLOCATE instead of COMMIT for frequently executed packages (greater than 20 per second). Be aware that RELEASE DEALLOCATE prevents some utilities from executing.
- ▶ Consider increasing the EDM pool size.

### ***LC24: pre-fetch scheduling***

This is latch type 56 (or X'38'). The latch is obtained when any prefetch engine is scheduled or ended, or a GETPAGE to check if a page is being prefetched. The user threads get the latch when scheduling a prefetch and when checking for a prefetch in progress. The prefetch engine gets the latch to remove itself from the chain when the prefetches are complete. High contention may occur with:

- ▶ Many dynamic prefetch in star joins
- ▶ CPU query parallelism with many degrees
- ▶ Small page sets scanned frequently by many concurrent threads.

Remedies for high values are:

- ▶ Reducing the many concurrent prefetches.
- ▶ Disabling dynamic prefetch if not required by setting VPSEQT=0. This may be possible if there is minimal real I/O and no need for parallelism. Prefetch engines for list and dynamic prefetch are still started even if all pages are resident in the pool. Sequential prefetch is not started until the first page miss occurs, but then continues until the prefetch operation is completed.
- ▶ Using more partitions, as there is one latch per data set.







## Part 2

# Application concurrency and lock optimization

Locking and concurrency are affected by options you choose when creating tables in DB2, by the design of your application programs, by options you choose when binding plans and packages, and the setting you have for some installation parameters, so you need to take all these into account throughout the development process.

In this part, we introduce DB2 9 serialization features and discuss application behavior with other concurrently running SQL data manipulation languages (DML), data definition language (DDL), data control language (DCL), and DB2 utilities and commands. We also describe lock optimization techniques that can be implemented in your database design and with subsystem parameters (DSNZPARMs) settings.

If you are going to access your DB2 data 24x7, then designing your application just for high performance batch or online access alone is not going to be enough. You should design your applications to be fail-proof with the assumption that batch jobs, online transactions, and some DB2 commands and utilities will run concurrently. The chapters in this part of the book will help you understand various locking scenarios you may encounter in your environment.

This part contains the following chapters:

- ▶ Chapter 4, “Database design considerations” on page 65
- ▶ Chapter 5, “Application design” on page 87
- ▶ Chapter 6, “Utilities, commands, and SQL” on page 161
- ▶ Chapter 7, “System considerations” on page 201





## Database design considerations

Physical database design is the outcome of the logical database design, and throughout the transformation from logical to physical database design, decisions have to be made that influence DB2 locking in addition to the overall performance. In this chapter, we discuss the impact of DB2 Data Definition Language (DDL) parameters on application concurrency.

A highly denormalized design results in the reduction of the number of tables in a system. Duplication of column values in different tables usually accompanies the reduction. With a denormalized design, we have to do more DB2 work for update or insert processes in the same logical unit of work. More DB2 work produces more DB2 locking in these situations. Thus, denormalization is not a good idea from a locking and concurrency perspective for update intensive applications. On the other hand, read-only operations are often simpler when we use a denormalized design (because many table joins can be avoided), resulting in less DB2 work and less DB2 locking.

Hot spots could result in significant lock waits for concurrent applications. During database design, potential hot spots have to be identified, and if you foresee locking problems with page level locking, it should be resolved at the table level by using row level locking, redistributing the clustering keys or through RANDOM order indexes.

Database designers should also keep in mind that any design ideas to improve the overall performance of the applications will have a positive impact on locking and concurrency. On the other hand, slow running applications have the tendency to hold the locks for a longer duration, which could potentially reduce concurrency.

Some important physical design implementation options for table spaces, tables, indexes, and other DB2 objects, such as sequences, triggers, and temporary tables, with a focus on locking and concurrency, are discussed in the following sections:

- ▶ Table space considerations
- ▶ Table design considerations
- ▶ Index design and locking considerations
- ▶ Instead of trigger
- ▶ Sequence object
- ▶ Database consideration
- ▶ Hot spot scenarios
- ▶ Database design recommendations for concurrency

## 4.1 Table space considerations

When creating or altering a table space, consider the following factors, because they affect DB2 locking:

- ▶ Type of table space (segmented, partitioned, or universal)
- ▶ Page size (BUFFERPOOL)
- ▶ Lock size (LOCKSIZE)
- ▶ Free space (FREEPAGE and PCTFREE)
- ▶ Maximum number of page or row locks (LOCKMAX)
- ▶ Maximum number of rows per page (MAXROWS)
- ▶ Data compression (COMPRESS)
- ▶ DEFINE parameter

### 4.1.1 Type of table space

The locking hierarchy of various table space types and the different lock sizes they can acquire are discussed in 2.1.1, “Lock size” on page 14 and in 4.1.3, “LOCKSIZE” on page 67. While creating a table space, you can choose the type of table space and it can have a considerable effect on locking and concurrency. The table space type cannot be changed using the ALTER statement.

#### Universal table space and partitioned table space

For a universal table space (as well as for a conventional partitioned table space), locks are always obtained at the partition level. Individual partitions are locked as they are accessed. Gross locks (S, U, or X) are also obtained on individual partitions only when needed.

You can specify the LOCKPART clause, but it has no effect. Starting with Version 8, DB2 treats all partitioned table spaces as though they were defined with LOCKPART YES. LOCKPART YES specifies the use of selective partition locking. When all the conditions for selective partition locking are met, DB2 locks only the partitions that are accessed. Selective partition locking in turn requires that a mass delete lock be taken to declare its presence on the table space (since selective partition locks may run without partition or page/row locks as long as lock avoidance is successful).

When the conditions for selective partition locking are not met, DB2 locks every partition of the table space.

If one of the following conditions is true, DB2 must lock all partitions:

- ▶ The plan is bound with ACQUIRE(ALLOCATE).
- ▶ The table space is defined with LOCKSIZE TABLESPACE.
- ▶ The LOCK TABLE statement is used without the PART option.

#### Segmented table space

In a segmented table space without partitions, rows from different tables are contained in different pages. Locking a page does not lock data from more than one table. Also, DB2 can acquire a table lock, which locks only the data from one specific table. Because a single row contains data from only one table, the effect of a row lock is the same as for a simple or partitioned table space; it locks one row of data from one table.

### **Simple table space**

DB2 no longer supports the creation of simple table spaces. However, an existing simple table space can contain more than one table. A lock on the table space locks all the data in every table. A single page of the table space can contain rows from every table. A lock on a page locks every row in the page, no matter what tables to which the data belongs. Thus, a lock needed to access data from one table can make data from other tables temporarily unavailable. That effect can be partly undone by using row locks instead of page locks.

### **LOB table space**

In a LOB table space, pages are not locked. Because the concept of rows does not exist in a LOB table space, rows are not locked. Instead, LOBs are locked.

### **XML table space**

In an XML table space, XML locks are acquired in addition to locks on pages and rows.

## **4.1.2 Page size**

The BUFFERPOOL parameter in the CREATE TABLESPACE statement determines the page size of the table space. For 4 KB, 8 KB, 16 KB and 32 KB page buffer pools, the page sizes are 4 KB, 8 KB, 16 KB, and 32 KB, respectively. If the BUFFERPOOL clause is not specified, the default buffer pool of the database is used (for LOBs, the DEFAULT BUFFER POOL FOR USER LOB DATA field on the installation window DSNTIP1 is used).

Using page locking in table spaces defined with 32 KB pages increases concurrency problems. Each page lock locks more rows in a 32 KB page than in a 4 KB page. We recommend that you use 4 KB pages for optimal locking and concurrency. Use 8 KB, 16 KB, or 32 KB pages only when you can justify defining rows longer than 4 KB.

## **4.1.3 LOCKSIZE**

The LOCKSIZE clause of the CREATE and ALTER TABLESPACE statements specifies the size for locks held on a table or table space by any application process that accesses it. The choices are ANY, ROW, PAGE, LOB, TABLE, or TABLESPACE. Having the ability to implement multiple levels of locking granularity significantly improves application concurrency without sacrificing the overall performance. Column LOCKRULE of the catalog table SYSIBM.SYSTABLESPACE contains the LOCKSIZE value.

The three levels in the hierarchy of lock sizes are discussed in 2.1.1, “Lock size” on page 14.

### **LOCKSIZE TABLESPACE**

A process acquires no table, page, row, LOB, or XML locks within the table space. That improves performance by reducing the number of locks maintained, but greatly inhibits concurrency. LOCKSIZE table space on a partitioned table space (including universal table space) implies a gross partition lock on all the partitions.

### **LOCKSIZE TABLE**

A process acquires table locks on tables in a segmented table space without partitions. If the table space contains more than one table, this option can provide acceptable concurrency with little extra cost in processor resources.

## LOCKSIZE PAGE

A process acquires page locks, plus table, partition, or table space locks of modes that permit page locks (IS, IX, or SIX). The effect is not absolute; a process can still acquire a table, partition, or table space lock of mode S or X, without page locks, if that is needed. In that case, the bind process issues a message warning that the lock size has been promoted.

## LOCKSIZE ROW

A process acquires row locks, plus table, partition, or table space locks of modes that permit row locks (IS, IX, or SIX). The effect is not absolute; a process can still acquire a table, partition, or table space lock of mode S or X, without row locks, if that is needed. In this case, the bind process issues a message warning that the lock size has been promoted or the user can get a message that locks have escalated.

## LOCKSIZE ANY

DB2 chooses the size of the lock; usually, DB2 picks 'page'.

## LOCKSIZE LOB

If a LOB must be accessed, a process acquires LOB locks and the necessary LOB table space locks (IS or IX). This option is valid only for LOB table spaces. The lock size cannot be a page or row for a LOB table space.

## LOCKSIZE XML

If XML must be accessed, a process acquires XML locks and the necessary XML table space locks (IS or IX). This option is valid only for XML table spaces. This option is not available in the CREATE or ALTER table space statements.

### 4.1.3.1 Altering LOCKSIZE

To alter LOCKSIZE, let S denote an SQL statement that refers to a table in the table space:

- The LOCKSIZE change affects S if S is prepared and executed after the change. This includes dynamic statements and static statements that are not bound because of VALIDATE(RUN).
- If the size specified by the new LOCKSIZE is greater than the size of the old LOCKSIZE, the change affects S if S is a static statement that is executed after the change.
- In all other cases, LOCKSIZE has no effect on S until S is rebound.

Table 4-1 lists the default lock size values for various types of table spaces.

Table 4-1 Default LOCKSIZE values for various table space types

Table space type	Lock size (default)
Implicitly created universal table spaces	ROW
LOB table spaces	LOB
Segmented table spaces <sup>a</sup>	ANY
Partitioned table spaces	
Simple table spaces (existing)	
Universal table spaces (explicitly created)	

a. If you omit NUMPARTS in the CREATE TABLESPACE, the table space created is segmented with a SEGSIZE of 4 and LOCKSIZE ROW, is not partitioned, and initially occupies one data set.

Before you choose LOCKSIZE TABLESPACE or LOCKSIZE TABLE, consider whether concurrency is likely to be affected. LOCKSIZE ROW increases concurrency. However, weigh this advantage against the CPU impact for locking.

### Row level versus page level locking trade-offs

The question of whether to use row or page locks depends on your data and your applications. If you are experiencing contention on data pages of a table space now defined with LOCKSIZE PAGE, consider LOCKSIZE ROW, but also consider the trade-offs.

The resource required to acquire, maintain, and release a row lock is about the same as that required for a page lock. If your data has 10 rows per page, a table space scan or an index scan can require nearly 10 times as much resource for row locks as that for page locks. But locking only a row at a time, rather than a page, might reduce the chance of contention with some other process by 90%, especially if access is random.

**Important:** Row level locking is not recommended for sequential processing.

Lock avoidance is very important when row locking is used. Therefore, use ISOLATION(CS) CURRENTDATA(NO) or ISOLATION(UR) whenever possible. In the case for ISOLATION(CS) CURRENTDATA(NO), DB2 may avoid acquiring a lock when reading data that is known to be committed. With ISOLATION(UR), DB2 does not care about commits and never gets a lock on pages or rows.

Lock avoidance techniques accomplish serialization by verifying the committed state of data without acquiring locks from IRLM. Thus, if only two of 10 rows on a page contain uncommitted data, DB2 must lock the entire page when using page locks, but might ask for locks on only the two rows when using row locks. Then, the resource required for row locks would be only twice as much, not 10 times as much, as that required for page locks. On the other hand, if two applications update the same rows of a page, and not in the same sequence, then row locking might even increase contention. With page locks, the second application to access the page must wait for the first to finish and might time out. With row locks, the two applications can access the same page simultaneously, and might deadlock while trying to access the same set of rows.

**Recommendation:** Set the LOCKSIZE of each table space to a level that is appropriate for most of your accesses. If a higher level of locking would be beneficial for a specific application, code a LOCK TABLE statement within that application.

## 4.1.4 LOCKMAX

The parameter LOCKMAX of CREATE TABLESPACE and ALTER TABLESPACE controls the maximum number of page or row locks that an application process can hold simultaneously in a single table space. In segmented table spaces, a single table can be locked. Segmented table space organization allows some tables to perform page or row locking and other tables in the same table space to perform table locking. If the number of locked pages or rows for a table exceeds the limit, a table lock is acquired for that table.

Lock escalation is controlled by the LOCKMAX option. If a process requests more than the number specified in LOCKMAX, the intent lock on the table space, partition, or table is promoted to S or X and the page or row child locks are released.

Table 4-2 shows the possible values of LOCKMAX.

*Table 4-2 Possible values of LOCKMAX*

LOCKMAX value	Meaning
SYSTEM	The value of field LOCKS PER TABLE(SPACE) on installation window DSNTIPJ determines the lock escalation threshold. If the value of the field is 0, lock escalation does not occur. If the value is n, where n > 0, lock escalation occurs as it does for LOCKMAX n.
0	Lock escalation does not occur.
1 to 2147483647	The maximum number of child locks (row, page, XML, or LOB) for the table or table space an application process can acquire before lock escalation occurs.

If LOCKMAX is omitted in the CREATE TABLESPACE or ALTER TABLESPACE statements, the default value of LOCKMAX depends on the specified value of LOCKSIZE. Table 4-3 summarizes the default value of LOCKMAX for various LOCKSIZE values.

*Table 4-3 Default value of LOCKMAX*

LOCKSIZE	Default LOCKMAX for CREATE TABLESPACE statement	Default LOCKMAX for ALTER TABLESPACE statement
<b>TABSPACE or TABLE</b>	0	0
<b>PAGE, ROW or LOB</b>	0	Unchanged
<b>ANY</b>	SYSTEM	SYSTEM

Lock escalation occurs when the number of locks held by a single process exceeds the LOCKMAX value on a table space. Lock escalation involves obtaining a table or table space lock, then releasing all of the page or row locks.

Lock escalation is a safety valve that DB2 provides in case an application overuses system resources. LOCKMAX 0 means that you are disabling this useful feature. Loss of concurrency is less likely to be incurred with lock escalation if the majority of the rows or pages in a table space are already locked, as long as RELEASE(COMMIT) is used. Lock escalation in such a situation reduces storage use, but more importantly, reduces the CPU time needed to traverse a lock hash synonym chain. Lock escalation also reduces the Internal Resource Lock Manager (IRLM) latch suspension time, which also reduces CPU time.

Lock escalation for a partitioned table space will cause DB2 to promote the intent locks for every partition where child locks are currently held. Any subsequent access to a new partition will cause DB2 to request gross locks on that partition from the outset. The LOCKMAX figure applies to all locks held at the table space level when it is a multi-table table space.

The column LOCKMAX on the catalog table SYSIBM.SYSTABLESPACE stores the maximum number of locks per user on the table or table space before escalating to the next locking level. A value of -1 indicates LOCKMAX=SYSTEM. Any escalation process is suspended during the execution of SQL statements for ALTER, CREATE, DROP, GRANT, and REVOKE.



Lock escalation is an expensive process, both in terms of lock management and concurrency. In today's high availability environment, you should design the application to allow no reason for lock escalations to occur. The benefit of lock escalation is in protecting DB2 from an excessive number of locks held in the IRLM on behalf of poorly designed applications that are not releasing their locks.

Lock escalation also prevents poorly designed applications from reaching the maximum number of locks per user specified by the NUMLKUS DSNZPARM. From an individual application perspective, this may seem like a good thing, but from an overall availability and performance standpoint, it is merely rewarding bad design. The focus should be on identifying and correcting those poorly designed applications that take an excessive number of locks.

For critical large table spaces that are accessed by many application processes concurrently, choose the LOCKMAX value so that there will not be any lock escalation within a unit of work, if NUMLKTS DSNZPARM is not large enough.

### 4.1.5 FREEPAGE and PCTFREE

You can use the PCTFREE and FREEPAGE clauses of the CREATE and ALTER TABLESPACE statement to define free space by reserving some empty pages within the table space (PCTFREE) and some empty space within a page (FREEPAGE), respectively. Sometimes it is possible to ease a locking problem by specifying more free space.

The FREEPAGE and PCTFREE parameters are honored by DB2 at LOAD time (LOAD and REORG utilities). In the case of an application that does only INSERTs into an initially empty table, DB2 will not honor the free space parameters.

Assuming that there is very little SQL INSERT activity, more free space means fewer rows per page, so fewer rows are locked by a page lock. However, there is a trade-off, as more free space can degrade scanning performance for clustered data due to scanning extra pages. If an SQL application accesses only a few rows on a page and uses page level locking, it is often better to use row level locking than to increase free space.

### 4.1.6 MAXROWS

The MAXROWS parameter specifies the maximum number of rows that DB2 will consider placing on each data page. This value is considered for insert operations, LOAD, and REORG. For LOAD and REORG the PCTFREE specification is considered before MAXROWS; therefore, fewer rows might be stored than the value you specify for MAXROWS. If you do not specify MAXROWS, the default number of rows is 255. With LOCKSIZE PAGE, if you want to get more granularity than page but less granularity than ROW (for example, a set of 10 rows), you can select an appropriate MAXROWS value. For smaller tables, a smaller value for MAXROWS (for example, 10) would improve the concurrency without increasing the impact to that of LOCKSIZE ROW.

In a data sharing environment, if you use MAXROWS=1 with LOCKSIZE PAGE, it would reduce the locking impact compared to LOCKSIZE ROW. See Chapter 11, "Monitoring data sharing locking activity" on page 365 for more information.

**Tip:** If you specify MAXROWS=1, then do not use COMPRESS YES.

### 4.1.7 COMPRESS YES

A table space contains more rows in a page when the data is compressed compared to when the data is not compressed. Consequently, DB2 uses fewer locks for a page scan on a table space when the data is compressed. However, there is less concurrency. Compressing data can result in a higher processing cost, depending on the actual SQL work load.

If you have concurrency problems with some of your table spaces when the data is compressed, consider using row locks.

### 4.1.8 DEFINE YES

If a partitioned table space is created with `DEFINE NO`, all partitions are also implicitly defined with `DEFINE NO`. Usually, the first data row that is inserted by a `LOAD` utility defines all data sets in the partitioned table space. If this process takes a long time, expect timeouts on the DBD.

Coding your `LOAD` job with `SHRLEVEL CHANGE` and using partition parallelism is equivalent to concurrent, independent insert jobs. For example, in a large partitioned table space that is created with `DEFINE NO`, the `LOAD` utility starts three tasks. The first task tries to insert the first row, which causes an update to the DBD. The other two tasks may time out while they wait to access the DBD. The first task holds the lock on the DBD while the data sets are defined for the table space.

If you are planning to run `LOAD` on individual partitions of a partitioned table spaces using partition parallelism, use `DEFINE YES` on the table space.

## 4.2 Table design considerations

When creating tables, you have no direct way to specify the locking mechanism. There are no keywords in the `CREATE TABLE` statement that directly influence locking except the `APPEND` option. However, referential integrity rules that are defined on tables can implicitly have a great impact on locking and concurrency. Other `CREATE TABLE` options, such as `VOLATILE`, `materialized-query-definition`, and the `CREATE GLOBAL TEMPORARY TABLE` and `DECLARE GLOBAL TEMPORARY TABLE` statements, can have some influence on locking and concurrency. Note that if you use an implicit table space, then the locksize is row by default and would need to be changed with `ALTER TABLESPACE`.

### 4.2.1 FOREIGN KEY references-clause

Each specification of the `FOREIGN KEY` clause defines a referential constraint. The `references clause` of a column definition also provides a shorthand method of defining a foreign key composed of a single column.

A delete operation on a parent table must acquire locks on the dependent tables. This locking can make those dependent tables less readily available for concurrent use. Locks on these table spaces are acquired only when they are used. Table spaces that are required to be accessed only for enforcing referential constraints are not affected by the `ACQUIRE(ALLOCATE)` option of the `BIND PLAN` command. How long the locks are held for referential integrity checking operations is also a performance consideration (for example, an application with frequent commits with `RELEASE(COMMIT)` may degrade the overall performance). With the `RELEASE(DEALLOCATE)` option, locks on these table spaces are released only when the plan terminates.

### 4.2.2 VOLATILE table

This option specifies that index access should be used on this table whenever possible for SQL operations.

Consider using volatile tables to reduce contention. DB2 will prefer an indexed access by applications when the table is defined as VOLATILE. Index access always returns the data in the same order and hence the probability of lock contention and deadlocks are significantly reduced. Thus, VOLATILE maximizes concurrency of operations in those cases.

**Note:** List prefetch and certain other optimization techniques are disabled when VOLATILE is used.

### 4.2.3 MQT instead of denormalization

Materialized Query Tables (MQTs) can also be used in place of denormalized tables. You can implement fully normalized tables to ensure data integrity during data modification and MQTs (simulating denormalized design) for efficient querying.

If MQT's base table's content changes frequently, then the refresh cost would be very high, and other issues, such as data currency and maintenance, may arise.

If the periodic refresh cost is less than the potential savings that can be achieved by using MQT, then you should consider using MQT in place of denormalizing the tables, if disk space cost can be justified.

### 4.2.4 NOT LOGGED facility

If you are performing large volumes of parallel INSERTs on to a table and experiencing high log write latch contention, then the NOT LOGGED option on the table could be used to improve the scalability. This feature could also be beneficial on MQTs and situations where data is being duplicated/replicated. Even though it is useful in certain situations, the NOT LOGGED option should be used only in those situations where loss of concurrency and recoverability is not a concern.

### 4.2.5 APPEND YES

The APPEND YES option specifies that data rows are to be placed into the table by disregarding the clustering during the insert and LOAD operations. This option would insert at the end of the table to minimize the cost of insertion by minimizing the read/write I/O, getpage, and number of locks.

Consider the scenario where the table space's LOCKSIZE is a page and an application program is trying to insert three rows into the same unit of work. If we have a clustering index and require free space in the data page, DB2 will try to find the optimal page in the clustering sequence and lock the page as it finds a suitable one. If all three pages are different, then DB2 would acquire three locks if APPEND NO option is used; it may acquire only one lock with APPEND YES if all three rows can be placed in the last page, as shown in Figure 4-1.

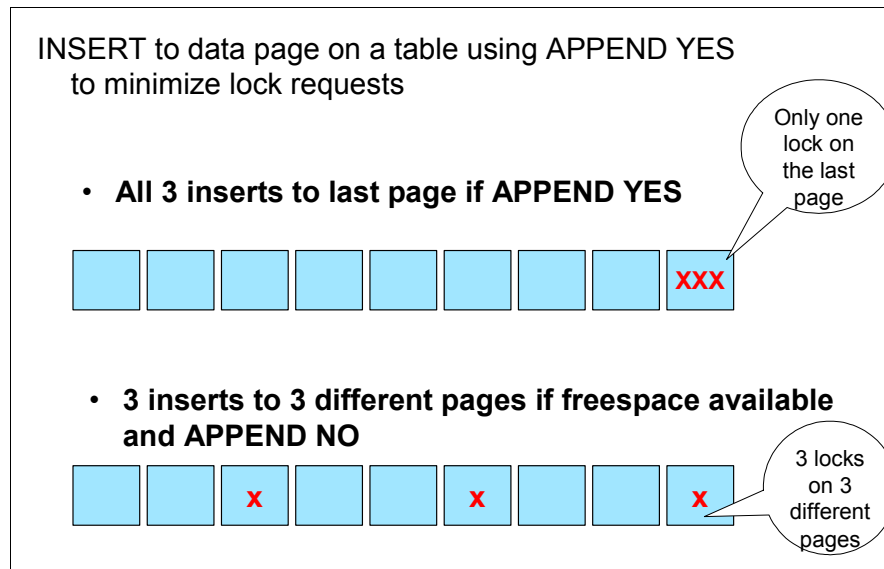


Figure 4-1 APPEND YES option to minimize number of locks on random INSERTs

## 4.2.6 Global temporary tables

Use global temporary tables when you need to store data for only the duration of an application process. Depending on whether you want to share the table definition, you can create a created temporary table or a declared temporary table. If your application needs a point in time or partial copy of a critical table, then using temporary tables to store pertinent data would relieve you of any need for locks on the critical table, thereby improving concurrency.

The two kinds of temporary tables are:

- ▶ Created temporary tables, which you define using a CREATE GLOBAL TEMPORARY TABLE statement
- ▶ Declared temporary tables, which you define using a DECLARE GLOBAL TEMPORARY TABLE statement

SQL statements that use temporary tables can run faster because of the following reasons:

- ▶ DB2 does no logging (for created temporary tables) and limited logging (for declared temporary tables).
- ▶ DB2 does no locking (for created temporary tables) and limited locking (for declared temporary tables).

For declared temporary tables, no row/page or table lock is acquired, but share level lock is acquired on the table space and DBD lock is acquired on the workfile database. Also, a segmented table lock is acquired when all the rows are deleted from the table or if the table is dropped.

## 4.3 Index design and locking considerations

There are no explicit transaction locks on index spaces. However, all index spaces on tables are implicitly locked at the same level as the table spaces containing those tables. P-locks are acquired by DB2 on index spaces in a data sharing environment.

During an insert, a lock on the next key is acquired to check for repeatable-read users, and an end-of-file index lock is established if there is a repeatable-read claim on a particular index.

The mass delete lock is also used to serialize uncommitted readers on indexes with a mass delete.

### 4.3.1 FREEPAGE and PCTFREE

FREEPAGE and PCTFREE are used to control free space in indexes. Index splits in random INSERTs can be reduced by providing a non-zero PCTFREE. A latch is held during index tree structure modification. Latch class 6 in statistics identifies index page splits.

Performance trace latch type 70 (X'46') shows contention in data sharing, and latch type 254 (X'FE') shows contention in non data sharing. These traces can be used to determine page splitting problems.

### 4.3.2 Page size

The BUFFERPOOL parameter in the CREATE INDEX statement determines the page size of the index.

Indexes can now have larger than 4 KB page sizes (that is, 8 KB, 16 KB, and 32 KB). More keys fit on a larger page before the page needs to split. Since the index leaf pages now split asymmetrically depending on the type of insert pattern on a page, larger page sizes allows optimal space usage on each page, effectively reducing latch contention on the index tree.

Larger page size on indexes also allows you to use the COMPRESS option, which is very useful where the indexes are created primarily for scan operations.

### 4.3.3 COMPRESS YES

An index can be compressed using the COMPRESS YES option. The buffer pool that is used to create the index must be 8 KB, 16 KB, or 32 KB in size (that is, higher than 4 KB). The physical page size on disk after compression will be 4 KB.

Index compression is recommended for applications that do sequential insert operations with few or no delete operations. Random inserts and random deletes can adversely affect performance with index compression. Index compression is also recommended for applications where the indexes are created primarily for scan operations. DB2 requires fewer getpages corresponding to the total number of leaf pages, when compressed, so the class 2 CPU time can be reduced.

If the indexes are kept in the buffer pool, there is no impact from compression.

In general, if the application performance improves, the lock duration decreases, and therefore the concurrency improves.

### 4.3.4 Data partitioned secondary indexes

Use data partitioned secondary indexes (DPSI) to promote partition independence and reduce lock contention and improve index availability, partition-level operations (such as rotating partitions), and recovery of indexes. This should also improve concurrency when you need to run your SQL processes concurrently with utilities accessing different partitions. Refer to 6.4.1, “Scenario with LOAD PART and concurrent SQL updates” on page 174 for more information.

### 4.3.5 CLUSTER

This option specifies whether the index is to be used as the clustering index of the table. This clause must not be specified for an index on an auxiliary table. CLUSTER cannot be specified if XMLPATTERN or key-expression is specified. The CLUSTER option can help with deadlock avoidance and hot spot scenarios.

### 4.3.6 RANDOM index

Index entries are arranged in a random order by the column. RANDOM cannot be specified in the following cases:

- ▶ For an index key column that is varying length in an index that is created with the NOT PADDED option
- ▶ With the GENERATE KEY USING clause
- ▶ If the index is part of the partitioning key

When index entries are put in a random order by the column, it could reduce lock contention, especially p-lock contention in a data sharing environment (this is one of the primary design points for this feature). Internally, DB2 “scrambles” the values in these key columns so they appear quasi-random. Randomized index values can still be used for equality look-ups and index-only access, but range scans are not supported.

Index contention, especially on a hot page, can be a major problem in a data sharing environment and can limit scalability. The randomized key order allows DB2 to spread out index keys within the whole index tree, instead of maintaining an ascending or descending order, thereby minimizing index page contention and turning a hot index page into a cool index page. See 4.6, “Hot spot scenarios” on page 80 for additional information.

A randomized index key can reduce lock contention, but can increase the number of getpages, lock requests, and read and write I/Os. Using a randomized index key can produce a dramatic improvement for a *hot spot* or degradation of performance. See Section *DB2 9 for z/OS Performance Topics*, SG24-7473 triggered action. For a list of allowable SQL statements, refer to the manual *DB2 Version 9.1 for z/OS SQL Reference*, SC18-9854. The statements are executed in the order in which they are specified. These statements will acquire the necessary locks as per the normal locking rules. Care must be taken to code these SQL statements in the same sequence as other concurrently running processes (accessing the same tables in the same order) to prevent deadlocks.

### 4.3.7 Instead of trigger

DB2 executes the triggered-action instead of the insert, update, or delete operation on the subject view. There is nothing special with respect to locking and concurrency while using “instead of triggers”; they just follow the base table locking rules.

## 4.4 Sequence object

If the locking problem was caused by an archaic number generation process with a single control table, then the sequence object can be utilized to solve such locking issues.

Locks kept too long may cause excessive waiting and even bottlenecks for hot pages. On the contrary, Figure 4-2 shows how locks released too soon may result in data integrity issues.

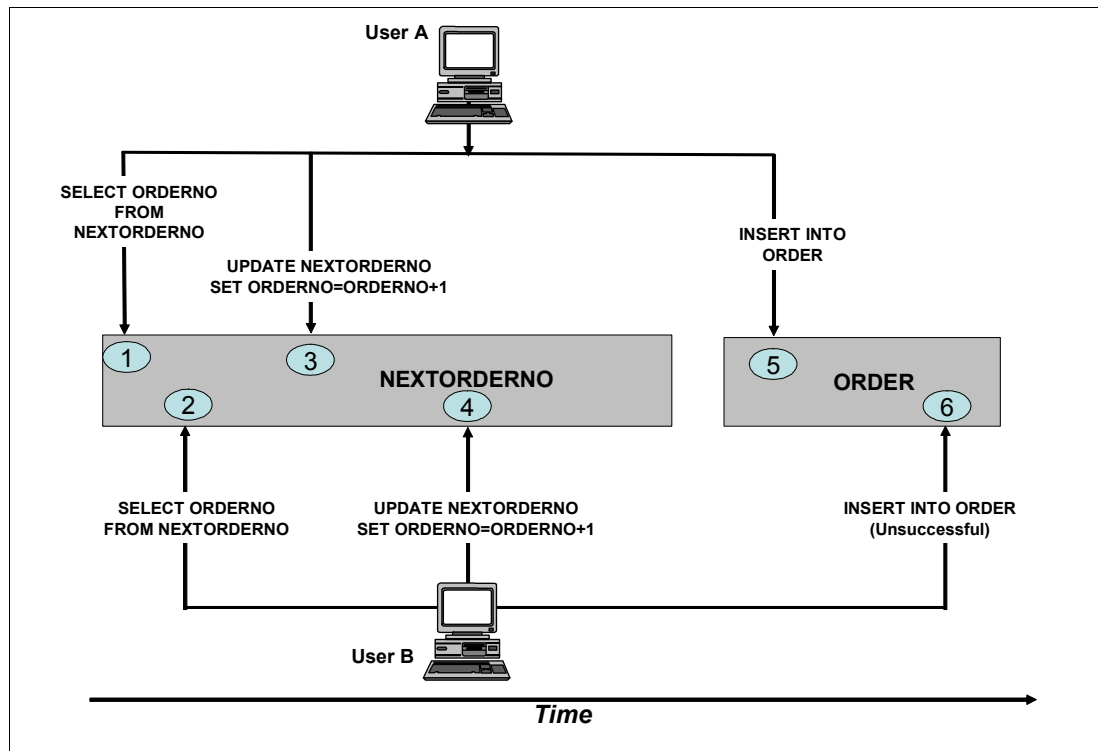


Figure 4-2 Lost update anomaly caused by releasing locks too soon

The steps outlined in Figure 4-2 are:

1. User A reads value 777 from table NEXTORDERNO using a singleton SELECT. With ISOLATION(CS) and lock avoidance, DB2 latches the page only during the time data is retrieved. If lock avoidance fails, DB2 acquires a lock as well, but this lock is released as soon as the SELECT is completed, if the isolation level is CS.
2. User B reads the same value (777) from table NEXTORDERNO using a singleton SELECT. DB2 again acquires only a latch or both a latch and a short S-lock, if the isolation level is CS.
3. User A changes value 777 to 778 using UPDATE. This process X-locks the page until the commit point.
4. User B tries to update NEXTORDERNO, but is suspended because of the X-lock held by user A.
5. User A inserts a new order 777 into the ORDER table and commits.
6. After updating the NEXTORDERNO to 779, user B tries to insert a new order 777 into the ORDER table. If there is no unique index on the order number column in the ORDER table, DB2 inserts an order 777 and the database is corrupted. If there is a unique index on the order number column in the ORDER table, the insert fails and the program must check for the SQL code and include the logic to handle the situation.

The conventional solution to avoid this problem is to use a more restrictive lock on the read operation. Instead of singleton SELECT, use SELECT defined with a cursor. Replace the SELECT ORDERNO FROM NEXTORDERNO (steps 1 and 2 in Figure 4-2 on page 77 above) with a FETCH referring to a cursor defined with SELECT ORDERNO FROM NEXTORDERNO FOR UPDATE OF. Close the cursor after the UPDATE.

This causes the following actions:

- ▶ User A acquires a U-lock at Step 1. This is promoted to an X-lock at step 3 and released at commit time.
- ▶ User B requesting a U-lock at step 2 waits until User A commits.

This solution prevents a deadlock with CS, RR, or RS isolation levels and requires the following steps in the application program:

1. Declare cursor and include the for update of clause.
2. Open cursor.
3. Fetch cursor.
4. Update using the where current of cursor clause.
5. Close cursor.
6. Insert.
7. Commit.

Another solution, offered by DB2 9, is to use SELECT FROM UPDATE where the ordeno is incremented and selected within one SQL statement.

Another possibility is to update the ordeno first, establishing an X lock, and then to go back and read the row to get the new value.

Using a sequence object to provide better number generation process without the impact of using a single control table would be a better solution to the above problem.

#### 4.4.1 Sequence objects

Sequences are complete, stand-alone objects and have no connection to a table. They can be used by multiple applications in different ways. A sequence is a stored object that simply generates the next ascending or descending value when requested by an application. They provide an excellent way for an application to obtain unique values for use in key structures.

There are two important considerations when using sequence objects:

- ▶ They may have gaps.
- ▶ In a data sharing environment, they may not be in strict sequential order.

These considerations also apply to identity columns.

##### **Generated values may have gaps**

There are various reasons why an application requesting a sequence number may not obtain the next sequential number. These are:

- ▶ A transaction advances a sequence and then rolls back.
- ▶ An SQL statement leading to generation of next value fails after the value is generated.



- ▶ NEXTVAL is used in a SELECT statement of cursor in DRDA, where the client uses block-fetch and not all retrieved rows are FETCHed.
- ▶ A sequence (or identity column) associated with sequence is altered and then ALTER rolled back.
- ▶ A sequence (or identity column) table DROPped and then DROP rolled back.
- ▶ A SYSIBM.SYSSEQ table space is stopped, leading to loss of unused cache values.
- ▶ A DB2 system failure or shutdown leading to a loss of unassigned cache values causes a gap in the sequence.

You need to be aware of these possibilities, which are possible in any environment (data sharing or otherwise).

### **Generated values may not be in strict sequential order**

In a data sharing environment, strict sequential order cannot be guaranteed when sequence numbers are cached on multiple members.

Before explaining how this task is possible, we discuss two keywords, CACHE/NO CACHE and ORDER/NO ORDER, that impact how the sequence object behaves:

- ▶ **CACHE/NO CACHE**

The CACHE/NO CACHE keywords specify whether or not sequenced values will be preallocated in memory. In a data sharing environment, each member has its own cache.

- ▶ **ORDER/NO ORDER**

The ORDER/NO ORDER keywords specify whether or not the sequence numbers must be generated in order of request.

Note that each member has its own cache and there is only one SYSIBM.SYSSEQUENCES table from which to request values. Assume an application is on two members of a 2-way data sharing system at the same time. If caching is active, each application can alternately request a sequence on each member and the sequence would be satisfied from that member's set of cached sequence numbers. For example, if CACHE is set to 20, each member would have a set of 20 cached values: member 1 could have cached 1 through 20 and member 2 could have cached 21 through 40. If the application alternated between members, the sequence order would be 1, 21, 2, 22, 3, 23, 4, and so on. These numbers are clearly not in sequence.

Table 4-4 summarizes the previous discussion and offers recommendations.

*Table 4-4 Sequence objects: impact of caching on order*

<b>Data sharing environment</b>	<b>ORDER keyword</b>	<b>CACHE keyword</b>	<b>Impact</b>
Data Sharing	ORDER	CACHE	The order is assured, but it may disable the caching of values (this is okay with a single application process).
		NO CACHE	The order is assured.
	NO ORDER	CACHE	The order is not assured (the application must be able to tolerate this scenario), and there is good performance.
		NO CACHE	The order is not assured (the application must be able to tolerate this scenario), and there is no reason to make a choice, because a performance improvement is possible.
Non-Data Sharing	ORDER	CACHE	DB2 changes CACHE to NO CACHE.
		NO CACHE	The order is assured and integrity dictates that you sacrifice performance.
	NO ORDER	CACHE	The order is not assured (the application must be able to tolerate this scenario), and there is good performance.
		NO CACHE	The order is not assured (the application must be able to tolerate this scenario), and there is no reason to make a choice, because a performance improvement is possible with CACHE.

## 4.5 Database consideration

Instead of assigning all your tables to one database, group the objects into separate databases based on your concurrency requirements. Plan for an adequate number of databases to reduce DBD locking if DDL, DCL, and utility execution is high for some of the objects in the same database.

You can set up utility jobs at the database level if the objects are grouped appropriately with your concurrency requirements in mind; this could essentially increase the availability of your DB2 data.

## 4.6 Hot spot scenarios

Hot spots are portions of a table or index that are continuously and repeatedly updated by the applications, for example, a counter or a total column. As a result, contention is likely to occur when accessing that data, and updates must serialize.

To avoid hot spots, you have to spread updates across the table or index by reducing the counter or total scope. For example, if you have a sales total in your table, every sale transaction will update this total, which becomes a hot spot. You can spread the updates by using departmental sales total, branch sales total, or even employee sales total.

You experience a hot page when concurrent processes try to lock the same page, even if they are not processing the same row. You experience a hot row when concurrent processes try to lock the same row. Figure 4-3 shows the difference between a hot page and a hot row. You typically experience a hot row in an application that keeps track of the last number used in a DB2 table. Hot rows and hot pages can occur in any type of program that tries to acquire locks that are incompatible with locks acquired by concurrent programs or SQL statements. The approaches to solving these problems differ.

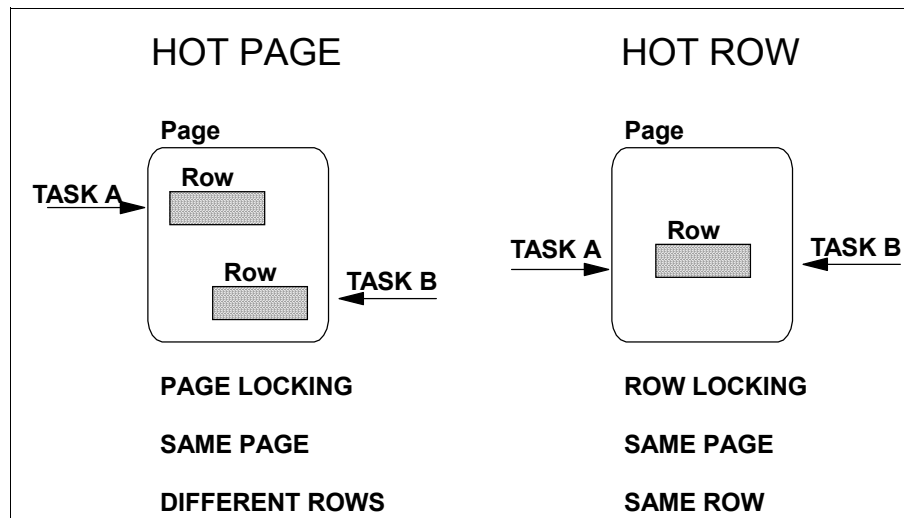


Figure 4-3 A hot page and a hot row

### 4.6.1 Hot data page problem

If you use page locking, a hot spot can occur on a data page. In this case, concurrent programs do not always process the same row. The hot data page problem typically occurs in the following cases:

- Small table

The probability of using the same page is very high because the number of pages is limited. The use of a table space scan, because the table space size is too small, is another factor that increases the probability of lock contention. Consider defining small tables with the VOLATILE attribute to ensure index access.

- Processing pattern

Although there may be many pages in the table, concurrent programs end up using the same limited set of pages.

- Data clustering sequence

If the clustering index correlates with the processing pattern, you increase the probability of a collision.

The possible solutions to this problem are:

- Use a smaller lock size or place fewer rows on each page.

Row level locking can be used to reduce lock contention when different processes are attempting to lock different rows on the same page.

The MAXROWS parameter can be used to reduce the number of rows on each page, and, with a value of 1, can obtain virtual row locking with LOCKSIZE PAGE.

Refer to “Row level versus page level locking trade-offs” on page 69 for more information about this topic.

- Change the processing pattern or data clustering sequence.

If you can modify your program logic to access DB2 data without concentrating on the same range, the contention decreases. You can also change your program's access pattern by altering the data clustering sequence, that is, redesigning the clustering index. Be aware of the impact of such a change on prefetch in your other applications.

## 4.6.2 Hot row problem

In the case of a hot row, concurrent processes try to access the same row, so you cannot resolve this problem by using a smaller lock size. Consider the following alternatives in resolving the hot row problem.

### Reduce lock duration

Attempt to reduce the duration of each lock on the hot row through the use of the techniques described throughout this chapter, such as:

- Avoid locks wherever possible.
- Acquire locks as late in the unit of work as possible.
- Release locks as soon as possible with commit.

### Evaluate the design

Hot rows are often created by a denormalization or an aggregation in the physical design of the tables. Review the design of the tables and the application to determine if a change might reduce or eliminate the hot row problem. The following case study describes how this can be done.

Assume your application needs to use a summary table to reduce the execution cost for read transactions to summarize the data, and the summary table is concurrently updated by transactions. Given the degree of concurrency, lock contention problems may occur.

Assume that you have an order history table, ORDER, and an order summary table, ORDER\_SUMMARY, that is updated every time a transaction updates the ORDER table. In this case, ordering transactions update both the ORDER table and ORDER\_SUMMARY table. Each insert, update, or delete in the ORDER table must be reflected in the totals of the ORDER\_SUMMARY table to provide data consistency. The ORDER\_SUMMARY table contains only one row, so you end up with a hot row problem, as shown in Figure 4-4 on page 83.

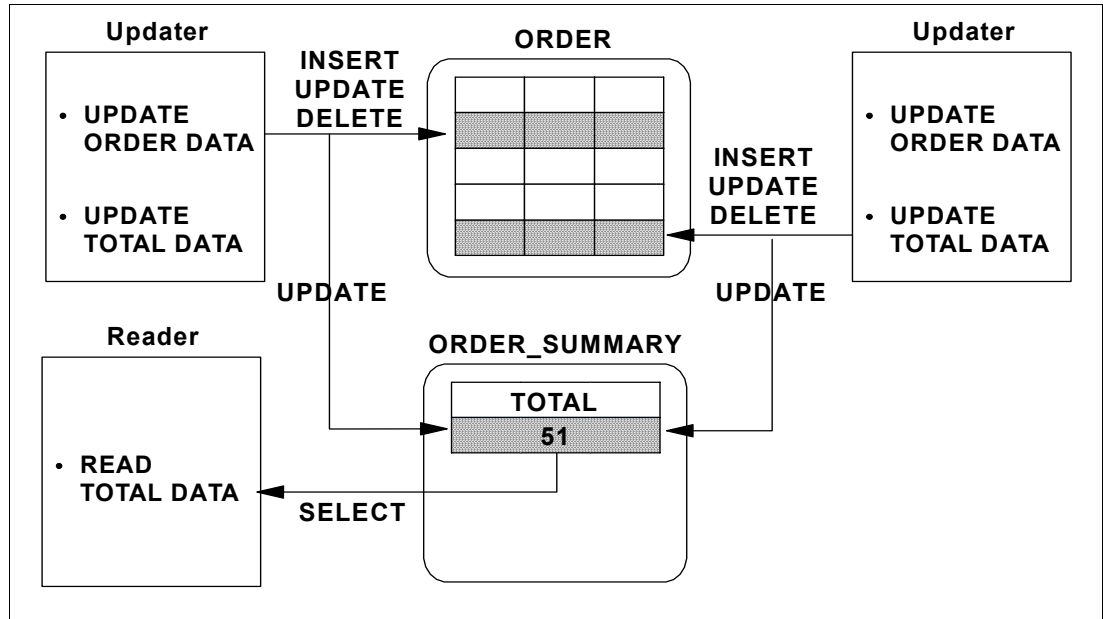


Figure 4-4 Updating a summary table

If you cannot reduce the application requirements, a solution to this problem may be to spread the updates among a larger number of rows in the **ORDER\_SUMMARY** table by adding another column to the primary key to achieve a more granular level of summarization, as shown in Figure 4-5.

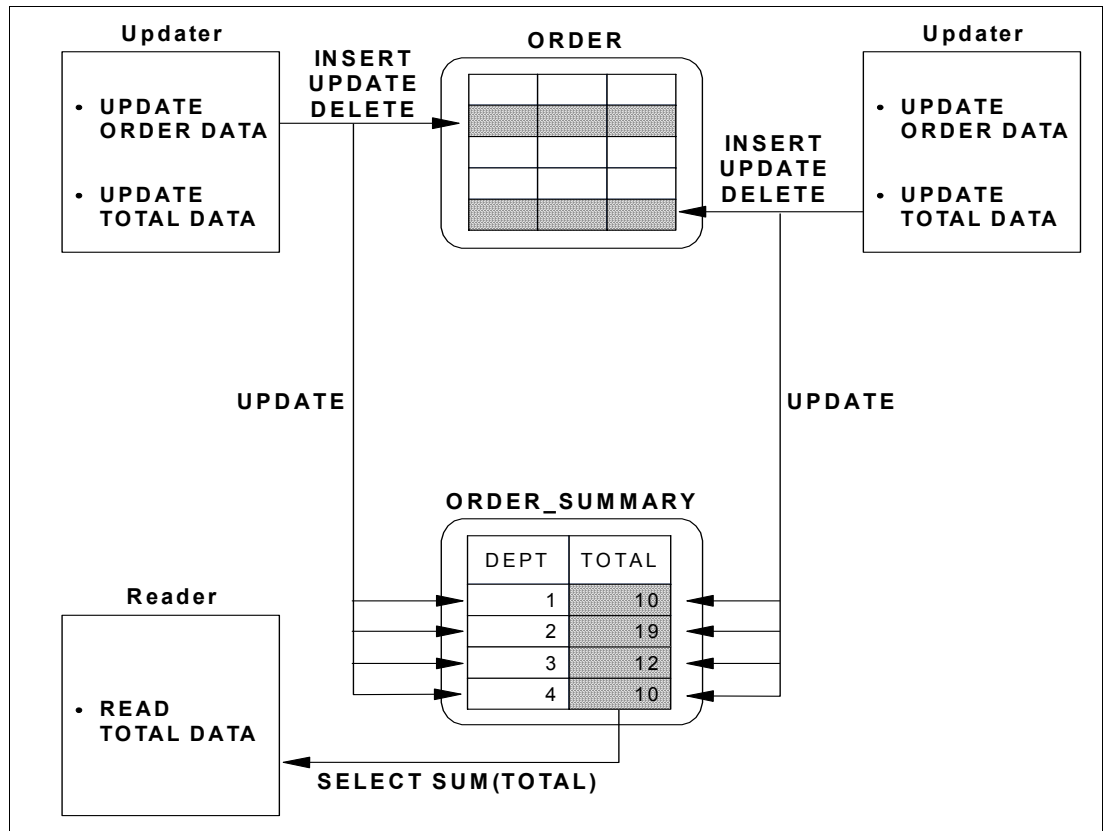


Figure 4-5 Spreading updates in a summary table

To improve concurrency, you can combine this technique of spreading updates with other techniques, such as a smaller lock size or reducing the number of rows per page.

This discussion applies if the updates on both tables are done in the same unit of work. However, if your business allows a delay in the propagation of the changes from the ORDER table to the ORDER\_SUMMARY table, a better solution is to use an asynchronous update technique such as Q replication, which is provided by IBM InfoSphere™ Replication Server products. Using Q replication, you can propagate updates asynchronously to the ORDER\_SUMMARY table. The performance of the transactions that change the ORDER table is not affected. There is no need for any additional programming to accomplish the replication.

### 4.6.3 Sequential numbers in DB2 tables

Another potential hot-spot scenario is when a table is used to assign the next sequential number. The best approach to solving hot spots in this case would be to use DB2 SEQUENCE objects. See 4.4, “Sequence object” on page 77 for more information.

## 4.7 Database design recommendations for concurrency

The following general recommendations and best practices for database design can help ensure improved concurrency on your DB2 system. This also provides a summary of various techniques recommended in this chapter.

To design your database to promote concurrency:

1. Keep like things together:
  - Give each application process that creates private tables a private database.
  - Put tables relevant to the same application into the same database.
  - Put tables together in a segmented table space if they are similar in size and can be recovered together.
2. Use an adequate number of databases, schema, or authorization-ID qualifiers, and table spaces to keep unlike things apart. Concurrency and performance is improved for SQL data definition statements, GRANT statements, REVOKE statements, and utilities. For example, a general guideline is a maximum of 50 tables per database.
3. Plan for batch inserts. If your application does sequential batch insertions, excessive contention on the space map pages for the table space can occur.

This problem is especially apparent in data sharing, where contention on the space map means the added impact of page P-lock negotiation. For these types of applications, consider using the MEMBER CLUSTER option of CREATE TABLESPACE. This option causes DB2 to disregard the clustering index (or implicit clustering index) when assigning space for the SQL INSERT statement. You may also consider using the APPEND option on the TABLE to alleviate this problem.

4. Use LOCKSIZE ANY unless you have reason not to. LOCKSIZE ANY is the default for CREATE TABLESPACE.

It allows DB2 to choose the lock size, and DB2 usually chooses LOCKSIZE PAGE and LOCKMAX SYSTEM for non-LOB/non-XML table spaces. For LOB table spaces, DB2 chooses LOCKSIZE LOB and LOCKMAX SYSTEM. Similarly, for XML table spaces, DB2 chooses LOCKSIZE XML and LOCKMAX SYSTEM. You should use LOCKSIZE TABLESPACE or LOCKSIZE TABLE only for read-only table spaces or tables, or when concurrent access to the object is not needed. Before you choose LOCKSIZE ROW, you should estimate whether doing so increases the impact for locking, especially from P-locks in a data sharing environment, and weigh that against the increase in concurrency.

5. For small tables with high concurrency requirements, estimate the number of pages in the data and in the index. In this case, you can spread out your data to improve concurrency, or consider it a reason to use row locks. If the index entries are short or they have many duplicates, then the entire index can be one root page and a few leaf pages.

6. Partition large tables to take advantage of parallelism for online queries, batch jobs, and utilities.

When batch jobs are run in parallel and each job goes after different partitions, lock contention is reduced. In addition, in data sharing environments, the data sharing impact is reduced when applications that are running on different members go after different partitions.

Certain utility operations, such as LOAD and REORG, cannot take advantage of parallelism in partition-by-growth table spaces.

7. Partition secondary indexes. By using data partitioned secondary indexes (DPSIs), you can promote partition independence, reduce lock contention, and improve index availability, especially for utility processing, partition-level operations (such as dropping or rotating partitions), and recovery of indexes.

However, using data-partitioned secondary indexes does not always improve the performance of queries. For example, for a query with a predicate that references only the columns of a data-partitioned secondary index, DB2 must probe each partition of the index for values that satisfy the predicate if index access is chosen as the access path. Therefore, take into account data access patterns and maintenance practices when deciding to use a data partitioned secondary index. Replace a non-partitioned index with a partitioned index only if you will realize perceivable benefits, such as improved data or index availability, easier data or index maintenance, or improved performance.

8. Specify fewer rows of data per page. You can use the MAXROWS clause of CREATE or ALTER TABLESPACE to specify the maximum number of rows that can be on a page. For example, if you use MAXROWS 1, each row occupies a whole page, and you confine a page lock to a single row. Consider this option if you have a reason to avoid using row locking, such as in a data sharing environment where the row locking impact can be greater.
9. If multiple applications access the same table, consider defining the table as VOLATILE. DB2 uses index access whenever possible for volatile tables, even if index access does not appear to be the most efficient access method because of volatile statistics. Because each application generally accesses the rows in the table in the same order, lock contention can be reduced.
10. Consider using randomized index key columns. In a data sharing environment, you can use randomized index key columns to reduce locking contention at the possible cost of more CPU usage, from increased locking and getpage operations, and more index page read and write I/Os.

This technique is effective for reducing contention on certain types of equality predicates. For example, if you create an index on a time stamp column, where the time stamp is always filled with the current time, every insertion on the index would be the greatest value and cause contention on the page at the end of the index. An index on a column of sequential values, such as invoice numbers, causes similar contention, especially in heavy transaction workload environments. In each case, using the RANDOM index order causes the values to be stored at random places in the index tree, and reduces the chance that consecutive insertions hit the same page and cause contention. This is beneficial in case of index hot spots and latch contention in data sharing environments, as described in the scenarios of 11.5.3, “High INSERT data scenario” on page 398.

Although the randomized index can relieve contention problems for sets of similar or sequential values, it does not help with identical values. Identical values encode the same and each are inserted at the same place on the index tree.

Since identical values encode the same, the RR reader behavior will not be affected even on a range of rows, provided we have matching index scan on all the random sequenced columns.

Example 4-1 shows the DDL for the RANDOM sequence index, which was used in the scenario of RR reader behavior with RANDOM sequence index.

*Example 4-1 RANDOM sequence index DDL*

---

```
CREATE UNIQUE INDEX RAVI.RNDXEPA1
ON RAVI.GLWTEPA_RANDOM
  (PROJ_NO          RANDOM,
   ACT_NO           ASC,
   EMP_NO           ASC)
USING STOGROUP GLWG01
PRIQTY 100 SECQTY 100
ERASE NO
FREEPAGE 0 PCTFREE 5
GBPCACHE CHANGED
CLUSTER
BUFFERPOOL BP16
CLOSE YES
COPY NO
PIECESIZE 2 G;
```

---

The RR reader SQL used in the scenario is shown in Example 4-2.

*Example 4-2 RR reader: sample SQL used*

---

```
SELECT *
FROM RAVI.GLWTEPA_RANDOM
WHERE PROJ_NO = 1
WITH RR
```

---

When you have an RR (repeatable read) reader holding a bunch of (say 35) row locks through index access with MATCHCOLS=1 on a three column index with the first column sequence defined as RANDOM, a concurrent INSERT on this table trying to insert outside this range of rows would work without any lock suspension, but any another INSERT into the range of qualifying rows would always will get suspended.

**Note:** Data sharing considerations are discussed separately in 12.1, “Database design considerations” on page 418.





# Application design

Concurrency is the ability of more than one application process to access the same data at essentially the same time. A lock associates DB2 data with an application process in a way that affects how other SQL processes can concurrently access the same data. It is not always necessary for a process to explicitly request a lock to ensure data integrity, as DB2 controls it for you. However, you can improve both performance and concurrency by understanding the effects of the parameters that DB2 uses to control locks.

Three main goals of concurrency to keep in mind during application design are:

- ▶ Minimize the number of locks requested within a unit of work.
- ▶ Minimize the duration of the locks acquired.
- ▶ Avoid acquiring more restrictive locks than what is required to accomplish the task.

In this chapter, you will learn various ways of minimizing the number of locks and lock durations in DB2 applications.

This chapter contains information about how to avoid application locking problems originating from concurrently running data manipulation processes and also describes how new DB2 features and a good application design can be combined to improve application concurrency with considerations to performance.

The following topics are covered:

- ▶ SQL design
- ▶ Optimistic concurrency control
- ▶ Using the SKIP LOCKED DATA option
- ▶ Unit of recovery and unit of work
- ▶ Lock demotion
- ▶ Locks acquired by unconventional SQL statements
- ▶ LOB and XML locks
- ▶ Access path influence on locking
- ▶ CICS: locking considerations
- ▶ Distributed application considerations
- ▶ Programming for the Instrumentation Facility Interface
- ▶ Performance considerations
- ▶ How to prevent locking problems

## 5.1 SQL design

A lock associates a DB2 resource with an SQL process in a way that affects how other SQL processes can be designed to access the same resource. The process associated with a resource is said to “hold” or “own” the lock. DB2 uses locks to ensure that no new process accesses uncommitted data that has been changed by another process. The mode of a lock dictates what kind of access to the locked DB2 resource is permitted to the lock owner and to any other concurrent processes. Whether any locks are acquired at all for an SQL statement and the mode of those locks depend on the following factors:

- ▶ Lock size of the target table space (refer to “LOCKSIZE” on page 67)
- ▶ The type of SQL processing being performed
  - SELECT with read-only or ambiguous cursor
  - INSERT ... VALUES(...) or INSERT ... fullselect
  - Searched UPDATE or DELETE (that is, without cursor)
  - SELECT with FOR UPDATE OF
  - Positioned UPDATE or DELETE (that is, with cursor)
  - Mass delete or TRUNCATE
  - LOCK TABLE statement
- ▶ The method of access to data (refer to 5.12, “Access path influence on locking” on page 138 for additional information)
- ▶ The isolation level of the plan, package, or SQL statement
- ▶ Whether the application uses SKIP LOCKED DATA option (refer to 5.7, “Using the SKIP LOCKED DATA option” on page 118)
- ▶ Type of concurrency control technique used (refer to 5.3, “Locking and concurrency techniques with isolation levels” on page 102)
- ▶ Other concurrent processes and whether or not the concurrent processes are running in different members of the data sharing group (refer to Table 10-6 on page 317)

### 5.1.1 Types of SQL processing and locks requested by DB2

DB2 application programs cannot directly lock a row or a page. However, based on the type of SQL statement, such as SELECT (that is, read-only), SELECT... FOR UPDATE (that is, positioned update/delete), searched update/delete, and so on, DB2 will automatically acquire the appropriate locks. Unlocking of rows/pages depends on what SQL statement was used to lock those rows/pages, the isolation level (for share locks), and when the application issues a COMMIT. Isolation CS holds share locks until DB2 has finished processing the row for the select, RS holds the share locks on all rows processed by the select until the SQL statement is completely finished, and RR holds share locks on all rows processed until a commit is issued. Isolation level UR holds no page or row locks and ignores any existing locks on pages or rows. All non-shared locks (that is, update and exclusive) are released at commit time.

The higher level locks (that is, table, table space, or partition locks) are released based on the RELEASE bind parameter; this is true for locks acquired for static SQL irrespective of how the higher level locks were acquired. For dynamic SQL, locks are released at commit, unless dynamic cache is used.

This section shows the locks requested by DB2 for some commonly used types of SQL statements. Each following sub-section describes the locking behavior for each of the three levels of LOCKSIZE values starting with the smallest size.

## LOCKSIZE= PAGE or ROW, TABLE, or TABLESPACE

Table 5-1 shows the mode of the locks requested during the processing of different types of SQL statements with the target table space's LOCKSIZE parameter set to either ANY (PAGE or ROW), TABLE, and TABLESPACE for various access path types.

Table 5-1 Possible lock modes for LOCKSIZE=ANY, PAGE, or ROW for various access paths

Type of processing	Isolation level <sup>a</sup>	Access path	Lock mode		
			Page or Row <sup>b</sup>	Table <sup>c</sup>	Table space <sup>d</sup>
SELECT read only or ambiguous cursor (including with HOLD option) <sup>e</sup>	CS <sup>f</sup>	Any	S <sup>f</sup>	IS	IS
	RS	Index, any use	S, U <sup>g</sup> , or X <sup>g</sup>	IS or IX <sup>g</sup>	IS or IX <sup>g</sup>
		Table space scan			
	RR	Index/data access			
		Index scan <sup>h</sup>			
		Table space scan <sup>h</sup>	None	S	IS or S
INSERT	-	-	X	IX	IX
Searched UPDATE or DELETE (without cursor)	CS	Index-only	X	IX	IX
		Table space scan and all other Index/data access	U <sup>i</sup> to X		
	RS	Index-only	X		
		Table space scan and all other Index/data access	(S <sup>j</sup> to X) or X		
	RR	Index-only	(S <sup>j</sup> to X) or X or None	IX or X or None	IX or X
		Index/data access	(S <sup>j</sup> to X) or None	X or None	
		Table space scan	None	X or None	
	SELECT with FOR UPDATE OF...(with a cursor)	CS	Any	U	IX
RS		Any	S <sup>j</sup> , U <sup>j</sup> or X <sup>g</sup>		
RR		Index/data access			
		Index scan <sup>h</sup>	S <sup>j</sup> , U <sup>j</sup> , X <sup>g</sup> or None	X, IX or None	X or IX
		Table space scan <sup>h</sup>	None	X or None	
Positioned UPDATE/DELETE (with a cursor)	CS, RS, RR	Only current row is involved	X or None	IX, X or None	X or IX

a. Isolation UR (uncommitted read) will not acquire page or row locks.

- b. Page and row level locks are always released at COMMIT time and higher-level locks are released based on the bind RELEASE parameter value. Page and row level locks are not acquired if there is a gross lock on the table, table space, or partition. These locks are taken on pages if LOCKSIZE is PAGE or on rows if LOCKSIZE is ROW. When the maximum number of locks per table space (LOCKMAX) is reached, locks escalate to a table lock for tables in a segmented table space or to a table space lock for tables in a non-segmented table space. Using LOCKMAX 0 in CREATE or ALTER TABLESPACE disables lock escalation.
- c. Used only for segmented table spaces. If the table space has a gross lock, then the corresponding tables will not have any locks.
- d. Includes partition locks, but not LOB table space locks. DB2 always try to use selective partition locking.
- e. If the table space is started for read-only access, DB2 attempts to acquire a gross S lock.
- f. If the package is bound with CURRENTDATA=NO, then DB2 might not acquire any lock (when lock avoidance check succeeds) on the row or page.
- g. If you use the WITH clause to specify the isolation as RR or RS, you can use the USE AND KEEP UPDATE LOCKS option to obtain and hold a U lock instead of an S lock, or you can use the USE AND KEEP EXCLUSIVE LOCKS option to obtain and hold an X lock instead of an S lock.
- h. Even if LOCKMAX is 0, the bind process can promote the lock size to TABLE or TABLESPACE. If that occurs, SQLCODE +806 is issued.
- i. The DSNZPARM, XLKUPDLT can be used to disable update locks for searched UPDATE and DELETE.
- j. If the DSNZPARM RRULOCK value is set to NO (the default), the lock mode would be S; otherwise it will be U or X, depending on the access path.

### ***Access path types used in Table 5-1 on page 89***

All access methods are either scan-based or probe-based. Scan-based means the index or table space is scanned for successive entries or rows. Probe-based means the index is searched for an entry as opposed to a range of entries, which a scan does. ROWIDs provide data probes to look for a single data row directly. The type of lock acquired sometimes depends on the access path used. From a locking perspective, the following access paths might be of interest and are included in the Table 5-1 on page 89:

- ▶ Index-only: The index alone is used to identify qualifying rows (and also to return the data on read-only statements).
- ▶ Table space scan: The data alone identifies qualifying rows and the return data (no index is used).
- ▶ Index/data access: The index is used or the index plus data are used to evaluate the predicate.
- ▶ Index, any use: The index is used to evaluate predicates (either Index-only or Index/data).

### **SELECT with read-only or ambiguous cursor**

For a SELECT with read-only or ambiguous cursor, with ISOLATION(CS) and CURRENTDATA(NO), lock avoidance is attempted for all rows (qualifying and nonqualifying), but with CURRENTDATA(YES), lock avoidance can occur for nonqualifying rows only. With CURRENTDATA(YES), the lock acquired on a qualifying row is held only until the next fetch.

When a lock avoidance check succeeds, low level (page or row) locks are not requested (thereby avoiding calls to IRLM), but DB2 takes a latch to ensure data consistency.

Even with lock avoidance, the higher level (table space or table) locks are acquired and held until the program deallocation or the next commit based on the value of the RELEASE bind parameter (for static SQL). If the table space is partitioned, partition level locks are not acquired with lock avoidance.

With ISOLATION RR and LOCKSIZE=PAGE or ROW, even if LOCKMAX is 0, the bind process can promote the lock size to TABLE or TABLESPACE when the access path is table space scan or index scan. If that occurs, SQLCODE +806 is issued.

If your application can tolerate incomplete or inconsistent results, you can also specify the SKIP LOCKED DATA option in your query to avoid lock wait times.

**Note:** On singleton SELECT statements in which no cursor is used, a lock is not held on the row or page unless WITH RS or WITH RR is specified on the statement.

When WITH HOLD is not specified, a commit operation commits all of the changes in the current unit of work and releases all row or page locks. For example, with a non-scrollable cursor, an initial FETCH statement is needed after a COMMIT statement to position the cursor on the row that follows the row that the cursor was positioned on before the commit operation.

When the cursor is defined as WITH HOLD, the lock duration changes based on the COMMIT and RELEASE parameters. The RELEASE parameter only applies to high level locks. The RELEASE impact on held cursors is that high level locks that may have been released at commit are held across commit. The position of a held cursor depends on its type:

- ▶ A non-scrollable cursor is positioned after the last retrieved row and before the next logical row. The next row is accessed with a FETCH NEXT statement.
- ▶ A static scrollable cursor that is held is positioned on the last retrieved row. The last retrieved row is accessed with a FETCH CURRENT statement.
- ▶ A dynamic scrollable cursor is positioned after the last retrieved row and before the next logical row. The next row is accessed with a FETCH NEXT statement. SQLCODE +231 is returned for a FETCH CURRENT statement.

No locks are acquired on declared temporary tables.

### ***The effect of possible materialization on cursors WITH HOLD***

If the cursor does not materialize on a workfile, the claims on the base DB2 objects held by the cursor WITH HOLD will be held until the first commit after the cursor is closed.

If the cursor does materialize, the claims on the base DB2 objects held by the cursor WITH HOLD will be released by the first commit following materialization (OPEN CURSOR). This is because the data from the base DB2 objects is read into a workfile when the OPEN CURSOR is executed. The cursor will continue to hold claims on the workfile, but all locks and claims on the base objects are released. Since utilities do not run on the workfiles, the claims on workfiles do not threaten concurrency and availability requirements.

### ***Acquiring X lock or U lock on a SELECT***

If you use the WITH clause to specify the isolation as RR or RS, you can use the USE AND KEEP EXCLUSIVE LOCKS, USE AND KEEP UPDATE LOCKS, and USE AND KEEP SHARE LOCKS options in SELECT and SELECT INTO statements.

The USE AND KEEP UPDATE LOCKS option obtains and holds a U lock instead of an S lock. The USE AND KEEP EXCLUSIVE LOCKS option obtains and holds an X lock instead of an S lock. Although requesting an UPDATE or EXCLUSIVE LOCK can reduce concurrency, it can also prevent some deadlocks, if used wisely.

## INSERT VALUES(...) or INSERT fullselect

Inserts never wait for an X-lock on the table pages, but tries another page if the best possible page is already locked. An insert lock is a conditional lock. If the lock on the preferred page (usually dictated by the clustering order) is currently held by another process, DB2 will find a different candidate page for the INSERT rather than wait for the lock on the page of first choice. See “Conditional versus unconditional” on page 45 for more information.

Inserts are thus important for locking purposes only if they suspend other programs doing updates or deletes (or some selects). They cannot be suspended themselves except when:

- ▶ The lock size is a table, table space, or table space partition due to either lock escalation or due to table space LOCKSIZE being TABLE or TABLESPACE.
- ▶ The UW first does a SELECT with RR, which gets an S table space or table lock and then the INSERT would promote the lock to SIX. This would be incompatible with all other transactions except those getting an IS lock on the TS or table.
- ▶ A SELECT with RR isolation (RR reader) is holding an S lock on a set of rows and the new row you are trying to insert would satisfy the RR reader’s selection criteria.

Example 5-2 shows the locks acquired by the INSERT statement shown in Example 5-1.

*Example 5-1 Sample INSERT statement*

---

```

INSERT INTO
    RAVI.GLWTEPA
VALUES (1, 1, 9
,16.06
,'1969-12-29'
,'2069-12-31'
, CURRENT TIMESTAMP
, 'THARUN'
, CURRENT TIMESTAMP
, 'TJ')

```

---

*Example 5-2 Lock information for a static INSERT ... VALUES(...) statement*

---

Type	Level	Resource		Number
----	-----	-----		-----
DPAG	S	DB=DSNDB06	PS=SYSUSER	1
	S	DB=DSNDB06	PS=SYSDBAUT	1
	S	DB=DSNDB06	PS=SYSDBASE	1
PSET	IS	DB=DSNDB06	PS=SYSUSER	1
	IS	DB=DSNDB06	PS=SYSDBAUT	1
	IS	DB=DSNDB06	PS=SYSDBASE	1
	IS	DB=RAVI	PS=GLWSPJA	1
	IX	DB=RAVI	PS=GLWSEPA	1
SKCT	S	Plan=RAVIPG9B		1
MDEL	S	DB=RAVI	PS=GLWSEMP	1
TABL	IS	DB=RAVI	PS=GLWSPJA	1
	IX	DB=RAVI	PS=GLWSEPA	1
ROW	X	DB=RAVI	PS=GLWSEPA	1
				-----
Total =				13

---

Table GLWTEPA belongs to table space GLWSEPA. Referential integrity is defined across five tables: GLWTDPT, GLWTEMP, GLWTPRJ, GLWTPJA, and GLWTEPA (belonging to the two table spaces GLWSEPA and GLWSEMP). Refer to “Stored procedures workload” on page 442 for details about the relationship across these tables.

A brief description of the Type of the lock displayed in Example 5-2 on page 92 is included in 5.20, “List of common lock types: OMEGAMON PE online monitor” on page 159.

**Note:** APAR PK96263 resolves some issues about displaying claims and resources correctly in this report.

Example 5-3 shows the output from DISPLAY DATABASE CLAIMERS statement.

*Example 5-3 Claimers while performing an INSERT*

NAME	TYPE	PART	STATUS	CONNID	CORRID	CLAIMINFO
GLWSEPA	TS		RW	TS0	DB2R3	(WR,C)
-			AGENT TOKEN 15046			
GLWSEPA	TS		RW	TS0	DB2R3	(CS,C)
-			AGENT TOKEN 15046			

The locks listed above are acquired on the object into which the insert is made. The fullselect acquires additional locks on the objects it reads, as though for SELECT with read-only cursor or ambiguous cursor, or with no cursor.

You cannot specify an isolation level for INSERT ... VALUES (...). But, for INSERT with fullselect, the isolation coded in the SQL applies to the fullselect only.

Example 5-4 shows all the locks acquired on an INSERT .... fullselect statement. The fullselect is on a segmented table space and the INSERT is into a universal table space with LOCKSIZE=ROW.

*Example 5-4 Sample INSERT .... fullselect statement*

```

INSERT INTO
RAVI.GLWTEPA_RANDOM
  SELECT EMP_NO , ACT_NO , 9
    ,EMPTIME
    ,EMSTDATE
    ,EMENDATE
    ,CREATED_TS
    ,CREATED_BY
    ,UPDATED_TS
    ,UPDATED_BY
FROM RAVI.GLWTEPA
FETCH FIRST 1 ROW ONLY
WITH CS

```

Example 5-5 shows the locks acquired on an INSERT ... fullselect statement. The first line in the output shows the IS lock on the table space (PSET) GLWSEPA. This table space contains the GLWTEPA table (which is used in the fullselect) and the table lock mode is also IS (as this is a segmented table space). There is also an entry with “PALK” as the lock type, which corresponds to the partition lock on the target table space (that is, DSN00345.GLWTEPAR) and an X lock on the row.

*Example 5-5 Locks acquired on a INSERT ... fullselect (1 row): OMEGAMON PE output*

Type	Level	Resource		Number
----	-----	-----		-----
PSET	IS	DB=RAVI	PS=GLWSEPA	1
	IS	DB=DSNDB06	PS=SYSOBJ	1
	IS	DB=DSNDB06	PS=SYSUSER	1
	IS	DB=DSNDB06	PS=SYSDBAUT	1
	IS	DB=DSNDB06	PS=SYSSTATS	1
	IS	DB=DSNDB06	PS=SYSDBASE	1
PALK	IX	DB=DSN00345	PS=GLWTEPAR	1
MDEL	S	DB=DSN00345	PS=GLWTEPAR	1
TABL	IS	DB=RAVI	PS=GLWSEPA	1
	IS	DB=DSNDB06	PS=SYSOBJ	1
	IS	DB=DSNDB06	PS=SYSSTATS	1
ROW	X	DB=DSN00345	PS=GLWTEPAR	1
				-----
Total =				12

### ***Locks on related tables during an INSERT***

If there is a referential integrity constraint defined on the target table (into which INSERT is being performed), DB2 has to check and make sure that the inserted value has a valid parent key. While performing the check, if any of the resources (row, page, table, table space, or partition) corresponding to the parent key is locked exclusively by another process, then DB2 will wait till this incompatible lock is released. If the incompatible lock is not released within the timeout interval, then the INSERT will get a timeout error.

### **Searched UPDATE or DELETE (without a cursor)**

With ISOLATION CS, DB2 will acquire a U lock (a read for update lock) on each of the qualifying rows. With additional non-indexable predicates, DB2 will NOT acquire X lock on each row touched (until it applies all non-indexable predicates) because DB2 does not know if the row will actually qualify for a DELETE/UPDATE until it applies all the non-indexable predicates. If the non-indexable predicates evaluate to false, then the U-lock will be released for that row and the internal processing moves to the next row. If the non-indexable predicates evaluate to true, then the U-lock will be upgraded to X-lock and held until COMMIT. DSNZPARM XLKUPDLT can be used to disable update locks for searched UPDATE and DELETE.

With ISOLATION CS and index-only row qualification (as indicated by a value of Y for the INDEXONLY column on PLAN\_TABLE), DB2 will acquire an X lock on all the qualifying rows. Thus, if the row qualification is index-only, then there will not be any lock promotions from U to X. This could also eliminate potential lock suspensions that may be experienced otherwise.

With an index-only access path, some times you may get a gross X lock on the table or partition, particularly when DB2 determines that it is going to scan a majority of the index pages (such as with MATCHCOLS=0 when ISOLATION is RR or RS without any index screening predicates).



With ISOLATION CS with table space scan access, beware of potential gross locks due to lock escalation.

With ISOLATION RR, a table space scan access path may result in a lock promotion to a gross lock on the table or table space or partition.

With ISOLATION RS and RR, DB2 will request an S lock or U lock based on RRULOCK value initially and it will be upgraded to an X lock after applying all the non-indexable predicates (if any). The same statement would request an X lock if RRULOCK is YES and the XLKUPDT value is YES. If DSNZPARM RRULOCK is set to NO, then the initial lock will be requested in S mode (with manual duration) and will be immediately upgraded to X mode (for commit duration). Refer to 7.3.10, “RRULOCK: U LOCK FOR RR/RS isolation” on page 210 for more information about RRULOCK DSNZPARM behavior.

You can use the XLKUPDLT option on the DSNTIPI installation window to disable the U lock on searched UPDATE and DELETE statements. When you set XLKUPDLT=YES, DB2 will not issue a second lock request to upgrade the lock from U to X (exclusive lock) for each updated row. DB2 will get an X lock. This parameter is primarily beneficial in a data sharing environment. Refer to 7.3.11, “XLKUPDLT: X LOCK FOR SEARCHED Update or Delete” on page 210 for more information.

When you specify XLKUPDLT=NO, DB2 might use lock avoidance when scanning for qualifying rows. When a qualifying row is found, an S lock or a U lock is acquired (depending on the isolation level and DSNZPARM, RRULOCK) on the row.

SKIP LOCKED DATA can be specified on a searched update statement. It suggests that rows are skipped when incompatible locks are held on the row by other processes. These rows can belong to any accessed table that is specified in the statement. SKIP LOCKED DATA can be used only when isolation CS or RS is in effect and applies only to row level or page level locks. Refer to 2.2.1, “Skip locked data” on page 27 for more information.

### ***Locks on related tables during a DELETE***

Operations subject to referential constraints can require locks on related tables. For example, if you delete from a parent table, DB2 might delete rows from the dependent table as well. In that case, DB2 locks data in the dependent table as well as in the parent table. If your application uses triggers, any triggered SQL statements can cause additional locks to be acquired.

The same is true for any DML statement on rows that contain LOB or XML values that might require locks on the LOB or XML table space and possibly on LOB or XML values within that table space.

### **SELECT with FOR UPDATE OF (cursor)**

When ISOLATION(CS) is used, FETCH statements acquires ‘U’ lock on the row or page being fetched. At the time of actual update, this U lock will be upgraded to an X lock.

When ISOLATION RS or RR is used, the lock acquired on the row or page fetched would depend on the RRULOCK DSNZPARM setting and whether or not the USE and KEEP .... LOCKS option is used. For example, if RRULOCK is NO and the USE and KEEP ... LOCKS option is not used, DB2 will request an S lock on the qualifying page/row at the time of fetch.

With ISOLATION RR, when the access path chosen is a table space scan, DB2 acquires a gross lock and there will not be any lower level page/row locks on the fetched page or row.

For a SELECT with FOR UPDATE OF cursor, FETCH operations referencing the cursor acquire U or X locks rather than S locks when:

- ▶ The isolation level of the statement is cursor stability.
- ▶ The isolation level of the statement is repeatable read or read stability and field U LOCK FOR RR/RS on installation window DSNTIPI is set (RRULOCK=YES) to get U locks.
- ▶ The isolation level of the statement is repeatable read or read stability and USE AND KEEP EXCLUSIVE LOCKS or USE AND KEEP UPDATE LOCKS is specified in the SQL statement, and an X lock or a U lock, respectively, is acquired at fetch time.

## **Positioned UPDATE/DELETE**

When a positioned UPDATE statement is executed, the cursor must be open and positioned on a row or rowset of the result table (and the fetch should have occurred in the same unit of work as the UPDATE WHERE CURRENT OF statement).

- ▶ If the cursor is positioned on a single row, that row is updated after acquiring an X lock if the lock already held on that row/page is either S or U.
- ▶ If the cursor is positioned on a rowset, all rows corresponding to the rows of the current rowset are updated after acquiring X locks on the rowset (or corresponding pages, if LOCKSIZE=PAGE or ANY).

If you execute a COMMIT after fetching from a open cursor and then immediately try to perform an UPDATE WHERE CURRENT OF operation, it would fail with an SQLCODE of -508, even if the cursor was defined with a WITH HOLD clause.

For a positioned update and delete, if the required X locks are already acquired at the time of FETCH operation, DB2 will not reacquire the locks.

If there is a need to promote the lock during the update operation, then a corresponding new gross lock request is sent to IRLM.

## **Updated rows (dirty rows)**

Until a commit or rollback operation releases the lock(s), only the application process that performed the UPDATE can access the updated row, because an X lock is held by the application process on the updated row (or page). If LOBs are not updated, application processes that are running with uncommitted read can also access the updated rows and pages. The X lock(s) also prevents other application processes from performing incompatible operations on the table.

## **Multi-row processing considerations**

In multi-row processing, the mode of lock acquired and under what conditions the locks are released would be similar to that of the corresponding single-row processing counterparts, but the lock's duration is usually elongated in the case of multi-row processing.

With CURRENTDATA(YES) and ISOLATION(CS) on a read-only or ambiguous cursor, the page/row locks pertaining to the entire row set will be held until the next rowset is fetched. Contrast this option with the single row processing option, where the page/row locks are held until the next single row fetch. The multi-row processing option usually results in considerable CPU performance improvements, but the lock duration is increased and concurrency is decreased when lock avoidance is not taking place. If concurrency is of paramount importance, then you should carefully evaluate the lock duration implications on your application while choosing the ROWSET size; the smaller the ROWSET size, the better the concurrency.

With CURRENTDATA(NO) and ISOLATION(CS) on a read-only or ambiguous cursor, lock avoidance is successful when there are no concurrent update activities. If there are concurrent updates in progress, then locks are acquired only on the pages or rows for which a lock avoidance check fails.

When using ISOLATION(RS), DB2 releases locks that were acquired on stage 1 qualified rows, but which subsequently failed to qualify for stage 2 predicates at the next fetch of the cursor (next ROWSET). Thus, the duration of locks held on a stage 2 non-qualifying row/page is longer than that of the corresponding single row processing situation.

**Caution:** Multi-row update operations, such as multi-row inserts, positioned updates, and positioned deletes, have the potential of expanding the unit of work and can have an adverse effect on the concurrency.

## LOCKSIZE=TABLE<sup>1</sup>

Table 5-2 shows the locks requested during the processing of some commonly used SQL (DML) statements with the LOCKSIZE parameter in the table space definition set to TABLE. The lock modes shown are valid for all possible types of access paths.

Table 5-2 Possible lock modes for LOCKSIZE=TABLE irrespective of the access path

Type of processing	Isolation level	Requested lock mode <sup>a</sup>	
		Table space	Table
SELECT read only or ambiguous cursor (including WITH HOLD option)	CS	IS	S
	RS		
	RR		
	UR	None	None
INSERT ... VALUES(...) or INSERT ... fullselec	CS	IX	X
	RS		
	RR		
	UR <sup>b</sup>	Not allowed	Not allowed
Searched UPDATE or DELETE (without cursor)	CS	IX	X
	RS		
	RR		
	UR	Not allowed	Not allowed
SELECT with FOR UPDATE OF... (with cursor)	CS	IX	U
	RS	IS <sup>c</sup>	S <sup>c</sup>
	RR		
	UR	Not allowed	Not allowed

<sup>1</sup> Used only for segmented table spaces.

Positioned UPDATE or DELETE (with cursor) single/multi-row processing	CS	IX	X
	RS		
	RR		
	UR	Not allowed	Not allowed

- For LOCKSIZE=TABLE, the lock state cannot be anything less than table (no page or row level lock is acquired).
- If WITH UR is coded on the fullselect, then the insert fails with a negative SQLCODE (-104).
- Assuming RRLOCK DSNZPARM is set to the default NO; if RRLOCK=YES, then the table space lock mode will be IX and the table lock mode will be U.

Only gross locks are acquired at the table level when the LOCKSIZE is TABLE. For a segmented table space with single table, you should use LOCKSIZE = TABLESPACE instead.

DB2 will not process row or page level locks on table spaces defined with LOCKSIZE TABLE.

### SELECT with FOR UPDATE OF (cursor)

With isolation CS, the table lock requested would be in U mode and the corresponding table space lock mode would be in IX mode.

With isolation RR or RS, the value of the installation parameter RRLOCK will determine the type of lock acquired. If RRLOCK=NO (which is the default), the lock requested on fetch would be in S mode at the table level and IS mode at the table space level. If RRLOCK=YES, the lock requested would be in U mode at the table level and IX mode at the table space level.

### LOCKSIZE=TABLESPACE

A table space lock is the largest size lock and it controls the most data, that is, all the data in an entire table space. Table 5-3 shows the lock requested during the processing of a few common types of SQL (only DML) statements with LOCKSIZE set to TABLESPACE. The lock modes shown here are valid for all possible types of access paths. With this setting, for a partitioned table space (including universal table space), DB2 will not be able to use selective partition locking and will lock all the partitions of the partitioned table space. In case of a segmented table space with multiple tables, all the tables will be affected with this setting. Table 5-3 also includes the claim information for each type of SQL statement.

Table 5-3 Possible lock modes for LOCKSIZE=TABLESPACE irrespective of the access path

Type of processing	Isolation level	Lock mode of the table space <sup>a</sup>	claim type
SELECT read only or ambiguous cursor (including WITH HOLD option)	CS <sup>b</sup> , RS	S	CS
	RR	S	RR
	UR	No TS locks	CS
INSERT ... VALUES(...)	N/A <sup>c</sup>	X	WR, CS
INSERT ... fullselect (isolation can be coded for the fullselect only)	CS, RS, RR	X	WR, CS
	UR <sup>d</sup>	N/A	N/A
Searched UPDATE or DELETE (without cursor)	CS, RS, RR	X	WR, CS
	UR <sup>d</sup>	N/A	N/A

SELECT with FOR UPDATE OF...	CS	U	CS
	RS, RR	S <sup>e</sup> or U	CS
	UR <sup>d</sup>	N/A	N/A
Positioned UPDATE or DELETE (with cursor) applicable to both single and multi-row processing	CS, RS, RR	X	WR, CS
	UR <sup>d</sup>	N/A	N/A

- For LOCKSIZE=TABLESPACE, the lock state cannot be anything less than table space (no table or page or row level lock is acquired).
- For LOCKSIZE=TABLESPACE, the lock requested will be the same irrespective of the bind parameter value for CURRENTDATA (YES or NO) and there will not be any lock avoidance at the table space level.
- The WITH UR clause cannot be coded at the statement level, but the isolation bind parameter on package or plan can be anything.
- If WITH UR is coded on the fullselect, then the insert fails with a negative SQLCODE (-104).
- If the DSNZPARM, RRULOCK is set to YES, then the lock requested will be in U mode; otherwise, it will be in S mode.

**Tip:** Use LOCKSIZE TABLESPACE or LOCKSIZE TABLE only for read-only table spaces or tables, or when concurrent access to the object is not needed.

When the table space is segmented, and if it was started for read-only access, then DB2 would acquire an intent share (IS) lock on the table space and share (S) lock on the table when you try to read the table with a read-only or ambiguous cursor. This would be true regardless of the value of the CURRENTDATA bind parameter used.

If the read only or ambiguous cursor was defined with HOLD, then the bind RELEASE parameter value will determine the lock duration, as shown in Table 5-4.

*Table 5-4 Read-only or ambiguous cursor with HOLD and table space lock release behavior*

Scenario	LOCKINFO <sup>a</sup>	Release description
The bind parameter is set to RELEASE(COMMIT) with the cursor still open.	H-S,S,C	The lock will be freed at commit.
The bind parameter is set to RELEASE(COMMIT) with the cursor still open, but after executing COMMIT.	H-S,S,H	The lock will be freed when all the held cursors are closed.
The bind parameter is set to RELEASE(COMMIT) after the CLOSE cursor but before executing COMMIT.	H-S,S,C	The lock will be freed at commit.
The bind parameter is set to RELEASE(DEALLOCATE) with the cursor open or closed and irrespective of whether COMMIT is issued or not.	H-S,S,A	The lock will be freed at deallocation.

- See 9.1.1, "DB2 commands" on page 256 for a description of LOCKINFO.

## 5.1.2 Type of table spaces and locks acquired by DB2

The locking hierarchy of various table space types and the corresponding lock sizes that DB2 can acquire are discussed in 2.1.1, "Lock size" on page 14 and in 4.1.3, "LOCKSIZE" on page 67.

## 5.2 Optimizing concurrency and locking

The primary objective of tuning application locking is to reduce the overall lock duration while still maintaining the integrity of the data in the database and your application. By reducing the durations of locks held by one process, it is possible to reduce the amount of lock suspensions of all other processes accessing the same DB2 objects. There are four components to overall lock duration, which we discuss individually in the following sections.

### 5.2.1 When will the lock be acquired

The acquisition of DB2 locks when executing a DBRM that is bound into a plan is controlled by the ACQUIRE bind parameter. If ACQUIRE(ALLOCATE) is specified, all table and table space level locks required for SQL statements in the plan are acquired at thread allocation. If ACQUIRE(USE) is specified, the locks will be acquired at the time the statement that requires the lock is executed. For more information about the ACQUIRE bind parameter, refer to 6.6.3, “BIND command” on page 180.

For packages, DB2 always acquires locks at the execution of the statement that requires the lock. Therefore, the only thing your application can do to control the acquisition of locks is to position the SQL statements in your application in an order that acquires the most restrictive locks as late as possible within each unit of work.

As an example, Figure 5-1 shows the same series of statements being executed in two different sequences.

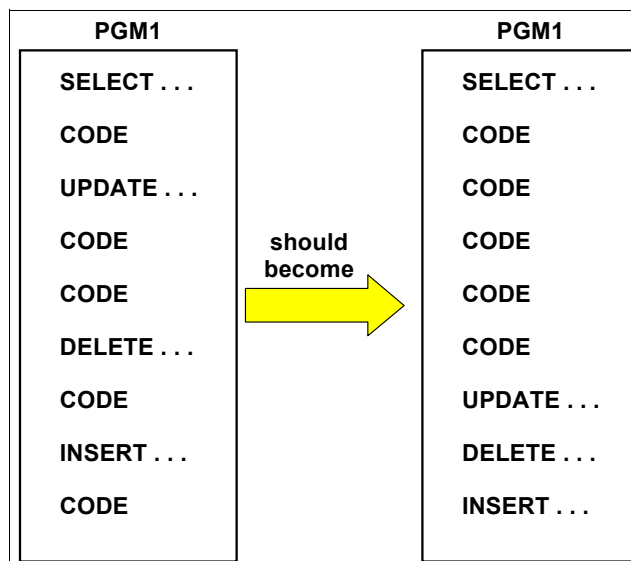


Figure 5-1 Changing a processing sequence to reduce lock duration

By arranging the sequence of statements so that the DB2 updates are performed at the end, a lock's duration can be reduced.

An additional benefit of this processing order might be that no database updates occur until all edits have been completed, which eliminates the need for a rollback in the event of an edit failure. Being able to maintain the same order of tables accessed in other programs can eliminate deadlocks.

## 5.2.2 Will a lock be required

A lock is always required for DELETE, INSERT, and UPDATE statements, so these question are only for the SELECT activity:

- Can this application afford to read uncommitted data?

If your application can afford to read uncommitted data (data that has been updated that will potentially be rolled back), then you can avoid taking locks for read activity by using an isolation level of UR. The benefits and dangers of using isolation UR are discussed in 5.5.1, “Read with no lock” on page 108.

- Can this application afford to have the row just read immediately updated?

Assuming that your application requires that each row read must be clean and committed, you next must consider whether your application cares if the row you just read is updated within your unit of work.

If the data read from the row is going to be used for a subsequent update, then you will want to tell DB2 to lock the page or row to prevent another process from updating the row in the middle of your unit of work. The recommended way to accomplish this is by using the SQL clause FOR UPDATE OF.

## 5.2.3 How restrictive will the lock be

In other words, how much of the table will you lock? You should ask the following questions:

- Can this application afford to have new rows inserted that qualify for the result set while it is being read?

If not, an isolation level of RR, or repeatable read, will lock all of the rows that qualify for the cursor and will not allow new rows that would qualify to be inserted. In 3.1, “Index specific lock” on page 42, we discuss this isolation level, but if your application really needs this level of locking, consider issuing a LOCK TABLE IN SHARE MODE before issuing your query.

- Can this application afford to have any part of the result set updated or deleted while it is being read?

If not, an isolation level of RS, or read stability, will lock all of the rows that qualify for the cursor.

If neither of these cases apply, then an isolation level of CS, or cursor stability, will probably satisfy your application’s locking requirements. Refer to 6.6.3, “BIND command” on page 180 for more information.

## 5.2.4 When will the lock be released

Exclusive locks will always be held until a COMMIT is issued, at which time they will always be released. S and U page and row locks taken by SELECT statements will also be released at COMMIT. However, if an isolation level of CS is used, S and U locks that were acquired when a row was fetched from a cursor are released as soon as the cursor moves to the next row or page.

The duration of table, table space, and partition locks are controlled by the RELEASE parameter on the bind. If RELEASE(COMMIT) is specified, then the locks are released at commit unless there are held cursors or held locators on the table, table space, or partition. If RELEASE(DEALLOCATE) is specified the table, table space and partition level locks are released at thread termination. For more information about the RELEASE bind parameter, refer to 6.6.3, “BIND command” on page 180.

## Duration of a partition lock

Partition locks follow the same rules as table space locks, and all partitions are held for the same duration. In a scenario when a package calls another package within the same unit of work, if one package is using RELEASE(COMMIT) and another package is using RELEASE(DEALLOCATE), all partitions use RELEASE(DEALLOCATE). A partition lock can be held past the commit point if it uses CURSOR WITH HOLD.

**Recommendation:** To reduce lock duration, favor the use of ISOLATION(CS) with CURRENTDATA(NO), RELEASE(COMMIT) and issue COMMIT statements at appropriate intervals.

## 5.3 Locking and concurrency techniques with isolation levels

The isolation clause is a fundamental part of DB2's locking protocol. The isolation clause is an attribute of the process executing the SQL statement. It governs the lock duration at the page or row level and the treatment of the result set of the SQL statement.

Based on the business requirements or toleration levels of the applications, the following locking and concurrency control scenarios can be achieved by coding the appropriate isolation level in the SQL statement or by choosing the appropriate value for the ISOLATION bind parameter on the plan or package (along with optional DSNZPARM settings).

ISOLATION(CS) lets DB2 release acquired row and page locks as soon as possible. CURRENTDATA(NO) lets DB2 avoid acquiring row and page locks as often as possible while reading with ISOLATION(CS). When you use ISOLATION(CS) and CURRENTDATA(NO), consider setting the SKIPUNCI subsystem parameter value to YES so that readers do not wait for the outcome of uncommitted inserts.

Table 5-5 shows the four isolation levels in order of decreasing preference for usage scenarios on locking and concurrency techniques used commonly in application design.

Table 5-5 Locking and concurrency techniques and isolation levels

Locking and concurrency technique	Isolation levels in order of decreasing concurrency ----->			
	WITH UR	WITH CS	WITH RS	WITH RR
Read with no lock on the table <sup>a</sup>	Yes	No	No	No
Optimistic concurrency control	Possible	Possible	No	No
SKIP LOCKED DATA	Not allowed	Allowed	Allowed	Not allowed
Searched update or delete <sup>b</sup>	No	Allowed	Allowed	Allowed
Skip uncommitted inserts	No	Yes, but only with DSNZPARM setting SKIPUNCI=YES	Yes, but only with DSNZPARM setting SKIPUNCI=YES	N/A
Use and keep ... locks <sup>c</sup>	No	No	Allowed	Allowed
Repeatable reads	No	No	Yes <sup>d</sup>	Yes

a. An inadvertently lost update is a possibility. A mass delete lock and claim are taken. In a data sharing environment, page-set P-locks are taken.

b. Any read with the intention of changing the data base (including INSERT from fullselect) cannot use isolation UR. If the isolation level is UR because of a BIND option, then the isolation level used by DB2 is CS.



- c. The locks could be EXCLUSIVE(X), UPDATE(U), or SHARE(S), and are kept until commit. The CLI attribute - Release locks at close cursor also applies to RS and RR, but the X/U/S manual duration locks are kept until commit (not close) when acquired as a result of the USE AND KEEP ... LOCKS clause.
- d. WITH RS, when you re-read, DB2 also returns newly qualifying rows (from insert/update) that were not returned during earlier reads.

### 5.3.1 Isolation levels

Isolation levels can be specified for individual SQL statements by coding the WITH clause in the SQL. If it is not specified at the statement level, then the ISOLATION bind parameter at the package level is used, and if the ISOLATION level is not provided at the package level, then the plan level ISOLATION is used.

**Important:** Use uncommitted read (WITH UR) on read-only SQL only if the application can tolerate uncommitted changed data, which may violate data integrity rules.

The WITH UR option may return uncommitted data. WITH UR is also called “read through locks” or “dirty read”. WITH UR SQL will ignore X locks on data pages, which indicate data has been modified but not yet committed, and return an unreliable value to the application. It could be either the uncommitted changed value or the “old” value.

The WITH CS or cursor stability option allows maximum concurrency with data integrity. The CURRENTDATA bind option could influence the locking behavior when you use this option.

The WITH RS or read stability option allows the application to read the same pages or rows more than once without allowing qualifying rows to be updated or deleted by another process. It offers possibly greater concurrency than repeatable read, because although other applications cannot change rows that are returned to the original application, they can insert new rows or update rows that did not satisfy the original search condition of the application.

The WITH RR or repeatable read option allows the application to read the same pages or rows more than once without allowing any update, insert, or delete operations by another process. All accessed rows or pages are locked, even if they do not satisfy the predicate.

**Note:** If incompatible isolation levels are attempted, DB2 will automatically try to correct it but without any warning. For example, if an update statement is executed as part of a package bound with isolation UR, then DB2 would automatically correct the isolation to CS.

#### Isolation level and page/row lock duration

DB2 allows application designers to control the trade-off between isolation and concurrency by specifying an isolation level. On static bind or dynamic prepare, different isolation levels affect the duration of locks. A lock can be avoided on ISO (UR) and on ISO(CS) with CURRENTDATA(NO). An isolation level can be specified at the SQL statement level as well.

There are four choices for the isolation level:

1. ISOLATION(CS), or Cursor Stability, acquires and releases page locks as pages are read and processed. CS provides the greatest level of concurrency at the expense of potentially different data being returned by the same cursor if it is processed twice during the same unit of work.
2. ISOLATION(RR), or Repeatable Read, holds page and row locks until a COMMIT point; no other program can modify or insert the data. If data is accessed twice during the unit of work, the same exact data will be returned.

3. ISOLATION(RS), or Read Stability, holds page and row locks until a COMMIT point, but other programs can INSERT new data. If data is accessed twice during the unit of work, new rows may be returned, but old rows will not have changed.
4. ISOLATION(UR), or Uncommitted Read, is also known as dirty read processing. UR avoids locking altogether, so data can be read that never was committed to the database.

Regardless of the ISOLATION level chosen, all page locks are released when a COMMIT is encountered. Figure 5-2 shows how long page locks are held by an application process fetching rows for isolation CS and isolation RS/RR in example 1 and 2, respectively.

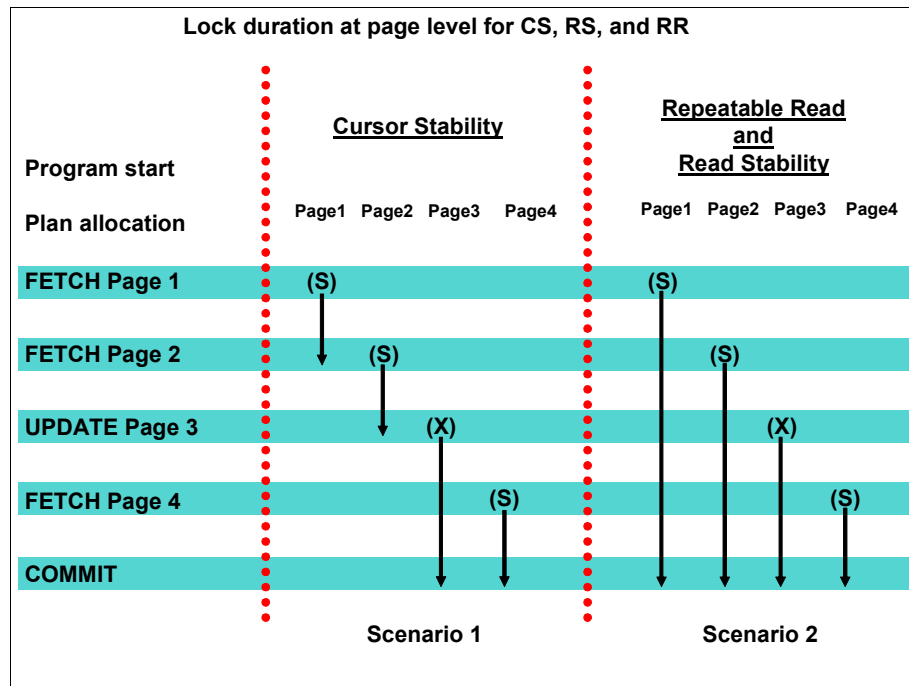


Figure 5-2 Page level lock duration for isolation CS versus RS/RR

The lock duration shown in Figure 5-2 are for page locks. If the LOCKSIZE=ROW, then the duration will be for row locks.

Scenario 1 in Figure 5-2 shows that with CURSOR STABILITY (CS), a page S lock, such as page 1, 2, and 4, is held only long enough to allow the cursor to move to another page. The same is true for U locks. Any X locks caused by an update, such as page 3, are always held until commit. In this situation, any page read can disappear and a new page can appear upon re-read. The user will not get dirty data and the user's changed data will not be seen by others (non UR) until the change is committed. This satisfies the CS requirement.

Scenario 2 in Figure 5-2 shows that with REPEATABLE READ (RR) as well as READ STABILITY (RS) index access, a page S lock is held for all accessed rows, qualifying or not, such as page 1, 2, and 4, until the next commit point. Any X locks, such as page 3, caused by an update, are always held until commit. The index access protects against any new rows to appear and the duration of the S lock (until commit time) protects against a read row from disappearing on re-read. If no index is used to access the data, a gross lock would have to be required to satisfy the semantics of RR.

**Note:** The SELECT...WITH ISO(RR or RS) USE AND KEEP EXCLUSIVE LOCKS option acquires X locks (at page/row level) on FETCH. Similarly, the USE AND KEEP UPDATE LOCKS option acquires U locks at the page/row level on FETCH.

## 5.4 Higher level locks

Higher level locks are locks taken on table, table space, or a partition of the table space. If gross locks are taken, it can override the DB2 rules for choosing initial lock attributes.

### 5.4.1 LOCK TABLE statement

The LOCK TABLE SQL statement has no special behavior when compared to LOCKSIZE TABLE or TABLESPACE settings. You can use LOCK TABLE on any table, including the auxiliary tables of LOB and XML table spaces. Refer to “LOCK TABLE statement” on page 145 for distributed environment considerations.

With the KEEP DYNAMIC option, under the conditions described in 5.14, “Distributed application considerations” on page 143, if a lock acquired through a dynamic LOCK TABLE statement is held past a commit point, DB2 demotes the lock state of a segmented table or a non segmented table space to an intent lock at the commit point.

Executing the statement requests a lock immediately, unless a suitable lock exists already. The bind option RELEASE determines when locks acquired by LOCK TABLE or LOCK TABLE with the PART option are released.

The LOCK TABLE statement is often appropriate for particularly high-priority applications. It can improve performance if LOCKMAX disables lock escalation or sets a high threshold for it.

Table 5-6 shows the modes of locks acquired in segmented and non segmented table spaces for the SHARE and EXCLUSIVE modes of LOCK TABLE. Auxiliary tables of LOB table spaces are considered non segmented table spaces and have the same locking behavior as universal table spaces.

The hybrid lock state of SIX is discussed in “Table space, partition, and table lock states for non-LOB or XML data” on page 16.

Table 5-6 Modes of locks acquired by LOCK TABLE

LOCK TABLE IN	Non segmented and universal table space	Segmented table space	
		Table	Table Space
EXCLUSIVE MODE	X	X	IX
SHARE MODE	S or SIX	S or SIX	IS
<b>Note:</b> The SIX lock is acquired if the process already holds an IX lock. SHARE MODE has no effect if the process already has a lock of mode SIX, U, or X			

## Recommendations for using LOCK TABLE

You can use LOCK TABLE to prevent other application processes from changing or reading any row in a table or partition that your process accesses. You have two options:

- ▶ LOCK TABLE in SHARE mode locks out changes and insertions of data.
- ▶ LOCK TABLE in EXCLUSIVE mode locks out reading, changes, and insertion of data.

For example, suppose that you access several tables. You can tolerate concurrent updates on all the tables except one; for that one, you need RR or RS isolation. You can handle the situation in several ways.

The LOCK TABLE statement locks out changes by any other process, giving the exceptional table a degree of isolation even more thorough than repeatable read. All tables in other table spaces are shared for concurrent update. If other tables exist in the same table space, the statement locks all tables in a simple table space, even though you name only one table. No other process can update the table space for the duration of the lock. If the lock is in exclusive mode, no other process can read the table space, unless that process is running with UR isolation.

You might want to lock a table or partition that is normally shared for any of the following reasons:

- ▶ Taking a “snapshot”: If you want to access an entire table throughout a unit of work as it was at a particular moment, you must lock out concurrent changes. If other processes can access the table, use LOCK TABLE IN SHARE MODE. (RR isolation is not enough; it locks out changes only from rows or pages you have already accessed.)
- ▶ Avoiding overhead: If you want to update a large part of a table, it can be more efficient to prevent concurrent access than to lock each page as it is updated and unlock it when it is committed. Use LOCK TABLE IN EXCLUSIVE MODE.
- ▶ Preventing timeouts: Your application has a high priority and must not risk timeouts from contention with other application processes. Depending on whether your application updates or not, use either LOCK IN EXCLUSIVE MODE or LOCK TABLE IN SHARE MODE. You can:
  - Bind the application plan with RR or RS isolation. But that affects all the tables you access and might reduce concurrency.
  - Design the application to use packages and access the exceptional table in only a few packages, and bind those packages with RR or RS isolation, and the plan with CS isolation. Only the tables accessed within those packages are accessed with RR or RS isolation.
  - Add the clause WITH RR or WITH RS to statements that must be executed with RR or RS isolation. Statements that do not use WITH are executed as specified by the bind option ISOLATION.
  - Bind the application plan with CS isolation and execute LOCK TABLE for the exceptional table. (If other tables exist in the same table space, refer to the caution that follows.) The LOCK TABLE statement locks out changes by any other process, giving the exceptional table a degree of isolation even more thorough than repeatable read. All tables in other table spaces are shared for concurrent update.

## 5.4.2 Lock demotion

Lock demotion is the process of changing the size or mode of a DB2 lock to a lower, less restrictive level.

To improve concurrency, gross locks (S, SIX, and X locks) that persist past commit for cached dynamic statements will be demoted at commit time. S locks will be demoted to IS, and SIX and X will be demoted to IX. However, demotion cannot occur if any of the following conditions exist:

- ▶ The table space has LOCKSIZE TABLESPACE.
- ▶ The segmented table has LOCKSIZE TABLE.
- ▶ The gross lock was acquired as a result of ISOLATION(RR).
- ▶ The gross lock was acquired because the table space was started for ACCESS(RO).
- ▶ The lock was held for static SQL.

Furthermore, for partitioned table spaces, lock demotion can only occur at the table space level, not the partition level. Lock demotion is tracked only at the table space and table level. In order to take advantage of this feature, statement caching must be enabled, the KEEP\_DYNAMIC bind parameter must be YES, and RELEASE(DEALLOCATE) must be specified on bind.

Also, IX locks on LOB table spaces will be demoted to IS if the locks must persist past commit (for held locators).

If there was a gross lock taken by a member of a global transaction, that lock is demoted to an intent lock at commit (because the global transaction dissolves into independent entities at commit and not all of them can have a gross X lock).

## 5.5 Read-only queries

In situations when lock avoidance is not possible, DB2 has to acquire locks for read-only queries. The read-only locks are shared, not exclusive, and it prevents others from changing the data while being read. Because an S-lock forces an updater to wait, Read-only SQL statements cannot be ignored in locking considerations.

When ISOLATION(CS) and CURRENTDATA(YES) are set, and if DB2 does not create a temporary table before bringing the rows to the program, a FETCH releases the lock when one of the following conditions is met:

- ▶ DB2 locks another data page or another row.
- ▶ End-of-file is reached.
- ▶ Cursor is closed.
- ▶ COMMIT is executed.

**Note:** A singleton SELECT with WITH CS will always try for lock avoidance irrespective of the value of CURRENTDATA parameter in the bind.

### 5.5.1 Read with no lock

No locks are acquired on table or table space if uncommitted read isolation (WITH UR) is specified. For all practical purposes, SELECT ... WITH UR (dirty read) is the closest we can get to no lock. Uncommitted read takes latches, but almost always never lock. However, there are restrictions on concurrent access, and in the following situation locks are acquired:

An uncommitted read operation makes a claim on the table space, so an application using UR isolation cannot run concurrently with a utility that drains all claim classes. Also, the application using isolation UR must acquire the following locks:

- ▶ A special mass delete lock is acquired in S mode on the target table or table space. A “mass delete” is a DELETE statement without a WHERE clause or a TRUNCATE statement; those operations must acquire the mass delete lock in X mode and thus cannot run concurrently.
- ▶ An IX lock on any table space used in the work file database (depends on the query). This prevents dropping of the work file table space while the application is running.
- ▶ If the global dynamic statement cache is not enabled (DSNZPARM, CACHEDYN=NO), then dynamic SQL readers using uncommitted read isolation take a DBD lock, which prevents operations such as DROP or ALTER, which accesses the same DBD.
- ▶ If XML values are read, XML locks and a lock on the XML table space appears. If the XML lock is not available because it is held by another application in an incompatible lock state, the UR reader skips the row and moves on to the next row that satisfies the query.
- ▶ In a data sharing environment, pageset p-locks are acquired to track inter-DB2 read write interest. Data sharing considerations with ISOLATION UR are discussed in Chapter 12, “Database and application design in data sharing” on page 417.

UR isolation is fast and causes little contention, but it reads uncommitted data. Do not use ISOLATION(UR) unless you are sure that your applications and users can accept the logically inconsistent data that can occur.

This introduces the possibility of two types of data inconsistency:

- Figure 5-3 shows the inconsistency that can occur when reading an updated row that is subsequently rolled back.

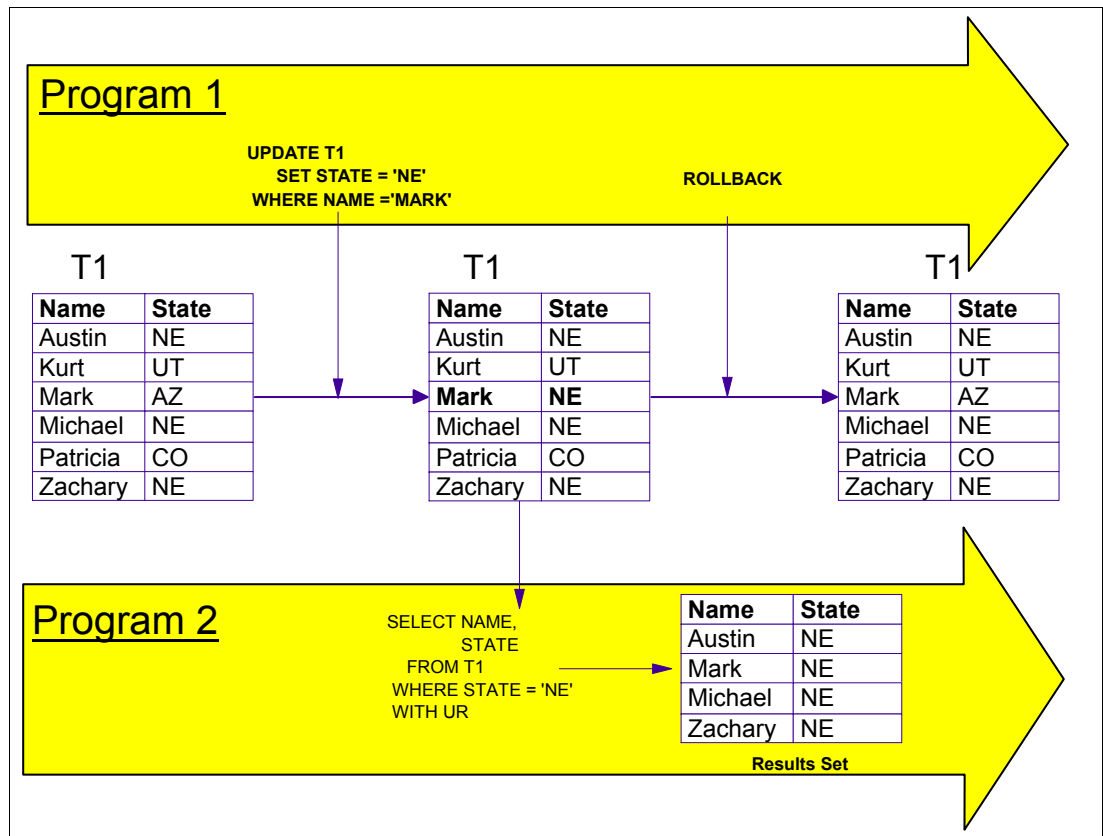


Figure 5-3 Data inconsistency caused by rollback

- Figure 5-4 on page 111 show an inconsistency that can occur when a row is qualified through the index using an updated column.

The index contains the new value of the column, but the data page still contains the old value. The query will return a DEPTNO value of 10, even though the query has a predicate that specifies that only rows with DEPTNO = 20 should be returned.



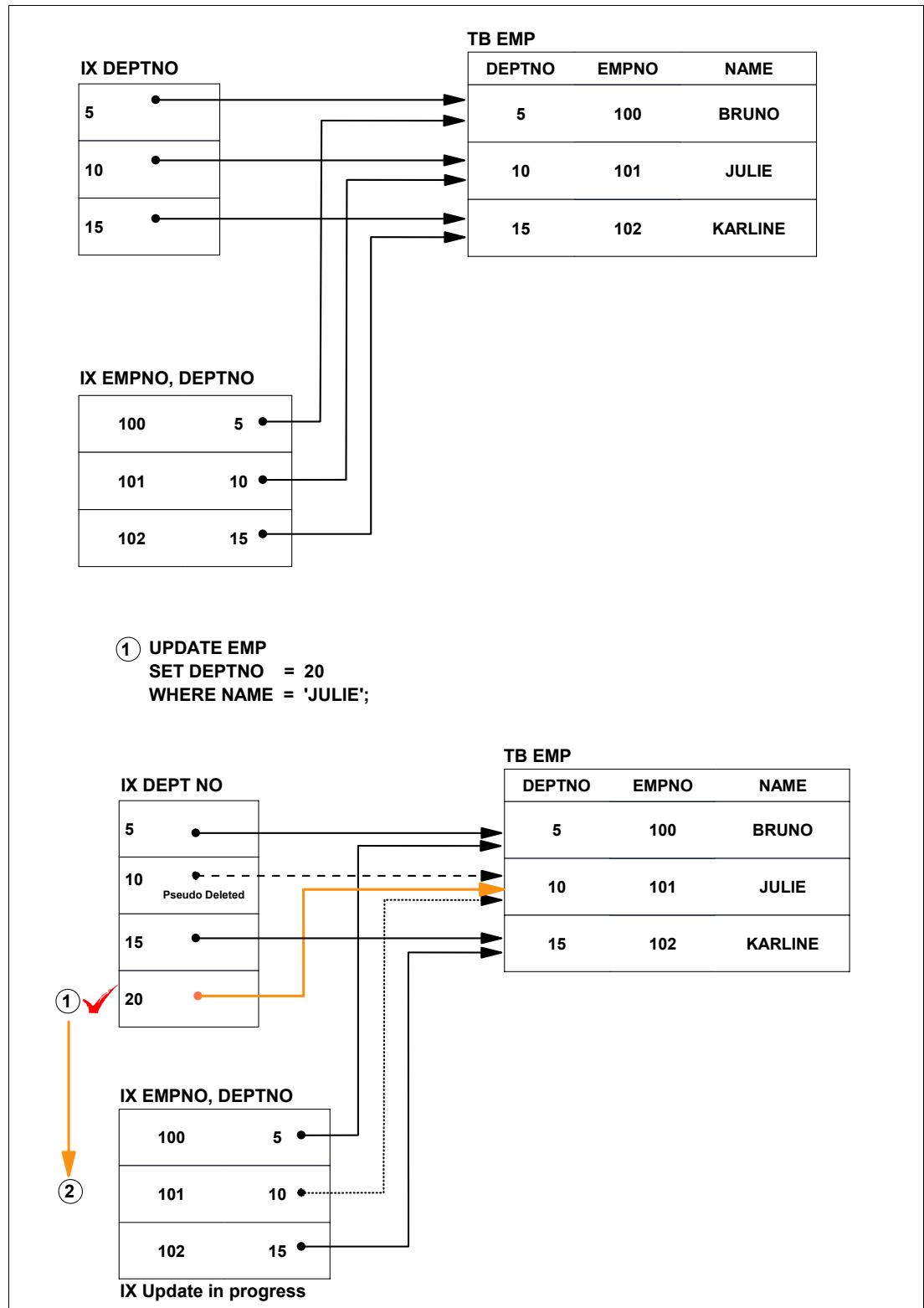


Figure 5-4 Data inconsistency between index and data page

Because of these potential data inconsistencies, you should only use uncommitted read when the inconsistencies will not adversely affect the integrity of your application. In particular, be wary of selects that qualify data using indexed columns that can be updated.

While UR can significantly improve throughput for applications that are experiencing a lot of lock contention, in the majority of cases there is little or no performance improvement using UR over using CS with CURRENTDATA(NO) with a large amount of lock avoidance.

Our recommendation is to only use UR on individual statements that are known to experience lock wait time and are not excluded from using UR by any of the data inconsistency issues described above.

If you use SPUFI with the AUTOCOMMIT NO setting to access production data and if you have the tendency to leave the data (result-set) in the window while performing other tasks, then you should consider executing a version of the SPUFI package that is bound with ISOLATION(UR), which executes the read-only queries without acquiring locks using the UR isolation level, and the updates execute using the CS isolation level. Thus, the query does not inadvertently hold locks on your production tables, which interferes with the production processes.

## 5.5.2 Mass delete locks

A mass delete lock is a special type of lock. The mass delete lock is used to serialize against transactions using the uncommitted read ISOLATION(SELECT ... WITH UR) option, which does not conflict with the table space or table lock.

Mass delete generally refers to a SQL DELETE statement without a WHERE clause, so all index entries and all rows are deleted. Mass delete locks are taken at the table space level (except for mass delete on segmented table spaces when the lock size is table) using the SQL statements shown in Table 5-7.

*Table 5-7 Sample SQL statements that use mass delete locks*

SQL statements	Mode of mass delete lock
DELETE FROM table1 (without WHERE clause)	X
TRUNCATE TABLE statement	X
EXCHANGE statement	X
INSERT (partitioned table)	S
SELECT ... WITH UR	S

If you are able to use TRUNCATE statement for mass delete operations (in place of the DELETE without WHERE clause), DB2 will still acquire mass delete locks on the table or table space. Refer to 5.9.3, "TRUNCATE statement" on page 130 for more information.

If an SQL statement encounters a timeout due to a mass delete lock being held by another concurrent process, then the timeout error message will show the table space name on which the mass delete lock conflict has occurred.

Example 5-6 shows the error message. Note that there is nothing explicit in the error message that indicates that the lock was a mass delete lock.

Example 5-6 Sample timeout error message on a SELECT with UR with concurrent mass delete

```

SQLCODE : -911                                DSNTIAR CODE : 0

DSNT408I SQLCODE = -911, ERROR:  THE CURRENT UNIT OF WORK HAS BEEN ROLLED BACK
        DUE TO DEADLOCK OR TIMEOUT.  REASON 00C9008E, TYPE OF RESOURCE
        00000200, AND RESOURCE NAME RAVIK  .GLWS001
DSNT418I SQLSTATE  = 40001 SQLSTATE RETURN CODE
DSNT415I SQLERRP   = DSNXRSFN SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD   = -190 13172746 13172878 13226969 -1026420734
        536870912 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD   = X'FFFFFF42' X'00C9000A' X'00C9008E' X'00C9D3D9'
        X'C2D21002' X'20000000' SQL DIAGNOSTIC INFORMATION

```

Example 5-7 shows the DISPLAY DATABASE command output when a mass delete is in progress on a segmented table space. Here, DB2 acquires an X lock at the table level and an IX lock at the table space level; in addition, a mass delete lock is acquired at the table level, which can only be seen through an online monitor like OMEGAMON PE.

Example 5-7 Sample DISPLAY DATABASE command output when a mass delete is in progress

```

DSNT360I -DB9A *****
DSNT361I -DB9A *   DISPLAY DATABASE SUMMARY
              *   GLOBAL LOCKS
DSNT360I -DB9A *****
DSNT362I -DB9A      DATABASE = RAVIK  STATUS = RW
              DBD LENGTH = 28256

DSNT397I -DB9A
NAME      TYPE PART  STATUS              CONNID   CORRID      LOCKINFO
-----
GLWS001   TS        RW                   TSO      DB2R3       H-IX,S,C
-          AGENT TOKEN 332
50        TB                   TSO      DB2R3       H-X,T,C
-          AGENT TOKEN 332

```

**Note:** Online utilities like REORG and RUNSTATS using the SHRLEVEL CHANGE option will also acquire mass delete locks in X mode.

## 5.6 Optimistic concurrency control

In most cases, it is best to let DB2 perform locking and serialization to ensure data integrity. Optimistic concurrency control is an alternative to database locking for concurrent data access. If only logically compatible processes are going to run concurrently, then there is no need for any kind of locking and serialization technique, and ideally we should be able to eliminate the locking overhead. But, practically speaking, we should be able to use a method that assumes that multiple processes running concurrently can successfully complete without affecting each other, and that therefore the individual processes can proceed without locking the tables that they affect. This method is widely used in applications and is called *optimistic concurrency control* (OCC). There are many flavors of OCC and we will discuss how OCC can be implemented using the new ROW CHANGE TIMESTAMP facility in DB2 and by exploiting DB2 lock avoidance techniques.

When process A, which has OCC implemented, reads data for an update, we are optimistic that if process B wants to update the data already read by process A, then process A will not update it, and that if other processes are not updating the data, then process A will update it. If this situation is true for the majority of occasions in your environment, then implementing OCC for this application is an alternative worth considering. If you have page level locking on your target table, then OCC is desirable if the processes are not going to conflict with each other.

If conflicts happen frequently, then the update would fail, you will have to reprocess the failed record, and the cost of reprocessing would render the optimistic locking pointless. Reprocessing would hurt overall performance significantly when compared to the performance savings achieved by avoiding the locks; other non-optimistic concurrency control methods would be desirable in such a conflict scenario.

If you already control concurrency by way of row level locking, then the optimistic concurrency control scheme discussed here would still be beneficial if the following conditions are true:

- ▶ The duration between read and update is longer than that of other concurrent incompatible processes.
- ▶ The applications that are running concurrently have higher business importance than that of the application designed for optimistic concurrency control.

In a row level locking scenario, we are optimistic that when process A, which has OCC implemented, reads a row for update, and if process B updates the same row, then process A will not update it, and that if others are not updating the row (they are just reading the row), then process A will most likely update that record.

Note that prior to DB2 9, the user maintained time stamp columns were used for validating whether or not the row read was changed by any concurrent processes at the time of actual update. In some cases, validation is being done by comparing all the selected column values to the corresponding values currently in the table at the time of update operation.

### **5.6.1 Optimistic locking using ROW CHANGE TIMESTAMP on the table**

In general, optimistic concurrency control is appropriate only for application processes that will not have concurrent updates on the same resource, such as information only (read-only) Web applications, single user applications, or pseudo-conversational OLTP applications, where the data will be read from the tables and presented to the users before performing the actual updates. Optimistic concurrency control is also appropriate for applications accessing tables defined with page level locking or higher level lock size when the concurrently running processes are accessing totally different sets of data.

When an application uses optimistic concurrency control, locks are obtained immediately before the read operation and released immediately after the read. The update locks are obtained immediately before an update operation and held until the end of the process.

#### **ROW CHANGE TIMESTAMP column type**

To safely implement optimistic concurrency control, you must establish a row change time stamp column with a CREATE TABLE statement or an ALTER TABLE statement. DB2 automatically maintains the contents of this column. This requires altering the table definition to add a new column, if the table to be used in optimistic concurrency control design already exists.

The column must be defined with one of the following null characteristics:

- NOT NULL GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
- NOT NULL GENERATED BY DEFAULT FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP

### ROW CHANGE TIMESTAMP expression

The ROW CHANGE TIMESTAMP expression specifies that a time stamp is returned that represents the last time when a row was changed. If the row has not been changed, the result is the time that the initial value was inserted. The result can be null. If the table has a ROW CHANGE TIMESTAMP column, it returns the value of the column. Otherwise, it returns a time stamp value derived from RBA/LRSN in the data page header.

### ROW CHANGE TOKEN expression

If the table has a ROW CHANGE TIMESTAMP column, then the ROW CHANGE TOKEN expression returns a BIGINT value derived from the time stamp value of the column. It will take the last 8 bytes of the DB2 time stamp and return as BIGINT. Otherwise, the BIGINT value will be derived from RBA or LRSN in the data page header. In a data sharing environment, it will be based on LRSN; otherwise, it will be based on RBA.

One of the underlying goals of optimistic concurrency control is to minimize the time for which a given resource is unavailable for use by other *compatible* processes. Figure 5-5 shows such an optimistic locking implementation scenario in an application.

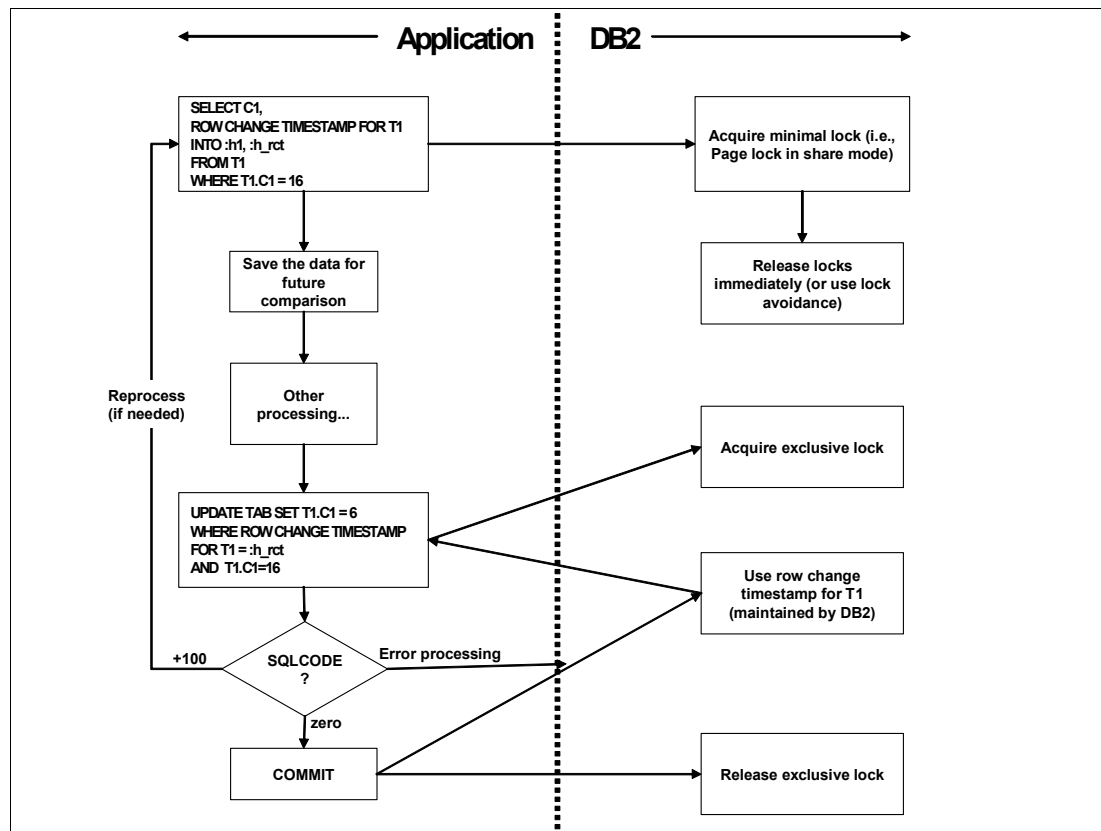


Figure 5-5 Sample optimistic locking implementation with ROW CHANGE TIMESTAMP

## Using ROW CHANGE TIMESTAMP on the page header

As of DB2 9 for z/OS, the applications have the option to use the ROW CHANGE TIMESTAMP or ROW CHANGE TOKEN option in the SQL code, but, if the target table is not altered to include the new GENERATED ALWAYS column of ROW CHANGE TIMESTAMP type, then DB2 returns a time stamp value derived from RBA/LRSN in the data page header. DB2 has a service task that records the new time stamp value and RBA value every 24 minutes. This service task stores only up to 2400 (RBA and time stamp) pairs for maximum of 40 days.

**Consideration:** The ROW CHANGE TIMESTAMP value on the page header is refreshed every 24 minutes and is not useful if the application has to check for updates more frequently than this refresh frequency.

## Optimistic concurrency control implementation example

One of the underlying goals of optimistic concurrency control is to minimize the time for which a given resource is unavailable for use by other compatible processes. Figure 5-5 on page 115 shows such an optimistic locking implementation in an application using the ROW CHANGE TIMESTAMP facility. The steps shown in Figure 5-5 on page 115 are explained as follows:

1. Acquire a minimal lock before the read to ensure data integrity. Be aware that a dirty read at this stage may increase the probability of failure of the update and may result in reprocessing of the affected records. The best option would be to read with ISOLATION(CS) and CURRENTDATA(NO) to get lock avoidance without sacrificing data integrity.
2. Select ROW CHANGE TIMESTAMP (or ROW CHANGE TOKEN) along with other pertinent information from the table (required columns and RID) by employing lock avoidance techniques.
3. Release the lock immediately after the read or employ lock avoidance techniques by using the ISOLATION (CS) with CURRENTDATA (NO) bind options.
4. Save the data, particularly the ROW CHANGE TIMESTAMP (or TOKEN) for future comparison.
5. Acquire the exclusive locks immediately before the update and hold on to it until the process ends or commits.
6. During the actual update, check whether the data read has been changed by another process since it was last read, by comparing the current row change time stamp (or token) with that of the saved values.
7. The update succeeds only when it is verified that the ROW CHANGE TIMESTAMP (or TOKEN) values match the ones in the RID; otherwise (in case another process has changed the value), the update fails with a return code of +100 (row not found for update).
8. The application has to reprocess the failed record, if needed.

Here is an example of SQL statements used in the optimistic concurrency control technique by way of ROW CHANGE TIMESTAMP in an application (also shown in Figure 5-5 on page 115):

```
SELECT C1, ROW CHANGE TIMESTAMP FOR T1
INTO :h1, :h_rct
FROM T1 WHERE T1.C1 = 16
```

You may also select the ROWID for better access path during update, if you are sure that ROWID will not change during the execution of your process.

The corresponding update statement looks like the following:

```
UPDATE T1 SET T1.C1 = 6
WHERE ROW CHANGE TIMESTAMP FOR T1 = :h_rct
AND T1.C1=16
```

Optionally, you can also make sure that the selected columns did not change using ROWID, if needed.

**Important:** Non-optimistic concurrency control (that is, SELECT with FOR UPDATE OF) methods would be desirable if conflicts could happen frequently (where there is a possibility of other concurrent processes trying to change the same data that you just read).

Do not run REORG with SHRLEVEL CHANGE when the application that employs optimistic concurrency control is running using ROWID to perform the row validity check during the actual update. A concurrently running online REORG has the potential to disrupt the optimistic concurrency control logic by reordering the ROWIDs.

Now let us analyze the effect of rollbacks, access paths, locksize, and isolation levels on the optimistic concurrency control mechanism:

- ▶ ISOLATION(CS): The best isolation level to choose for optimistic concurrency control implementation, as it can utilize lock avoidance technique using the CURRENTDATA(NO) option. Isolation UR may introduce data integrity problems. Isolation RR and RS will not work, as the row/page locks are not released until the next commit.
- ▶ LOCKSIZE: Both page level locking and row level locking would work fine. Higher level locks should be discouraged on tables used in the applications that implement optimistic concurrency control. For the same reason, care should be taken to avoid lock escalation, as lock escalation on the target table because of a concurrently running originally compatible process may contend with the update and may cause potential timeouts.
- ▶ Access path: Some seemingly compatible processes may not be compatible with optimistic concurrency control logic due to the influence of access paths on the lock size, such as the ones discussed in 5.12.3, “Table space scan with repeatable reads” on page 141.
- ▶ Rollbacks: If you implement OCC with dirty read, then any newly inserted records (that are not committed at the time of read) are visible during the read, and if they are rolled back, then they could cause the subsequent update to fail. This would not be an issue as long as dirty reads are avoided. So, ISOLATION(CS) with CURRENTDATA(NO) can ensure data integrity and at the same time reduce locking effectively.

## 5.6.2 Optimistic locking in updateable static scrollable cursors

Optimistic locking does not happen automatically. The application will need to select ROW CHANGE TIMESTAMP (or TOKEN) first, then, during an update, a predicate needs to be added as a condition to tell whether the row has been modified or not. The static scrollable cursor uses the optimistic locking technique automatically.

The technique used by DB2 with updateable static scrollable cursors has always been the optimistic locking technique. Serialization is accomplished by delaying locking acquisition to the time of actual update. No locks are obtained at the time of fetch. Instead, the column values returned to the application are saved. Eventually, on a positioned UPDATE or DELETE request, the corresponding row is locked and the current values are compared to the saved values. If the values match, the UPDATE or DELETE is allowed.

DB2 uses optimistic concurrency control to shorten the amount of time that locks are held in the following situations:

- ▶ Between consecutive fetch operations
- ▶ Between fetch operations and subsequent positioned update or delete operations

DB2 cannot use optimistic concurrency control for dynamic scrollable cursors. With dynamic scrollable cursors, the most recently fetched row or page from the base table remains locked to maintain position for a positioned update or delete.

Refer to the *DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide*, SC18-9851 for details about processing positioned update and delete operations with static scrollable cursors without optimistic concurrency control and with optimistic concurrency control.

## 5.7 Using the SKIP LOCKED DATA option

This option, when used in an SQL statement, specifies that the select-statement, searched UPDATE statement (including a searched update operation in a MERGE statement), or searched DELETE statement that is prepared in the PREPARE statement will skip rows when incompatible locks are held on the row by other processes (not all the data that is locked by other processes will be skipped, just the incompatible ones will be skipped). These rows can belong to any accessed table that is specified in the statement. SKIP LOCKED DATA can be used only when isolation CS or RS is in effect and applies only to row level or page level locks.

SKIP LOCKED DATA can be specified only in the searched UPDATE statement. SKIP LOCKED DATA is ignored if it is specified when the isolation level that is in effect is repeatable read (WITH RR) or uncommitted read (WITH UR).

A typical use for this option is in a queuing application. Generally, if your application logic requires discarding locked data or if it can tolerate incomplete or inconsistent results, you can specify the SKIP LOCKED DATA option in your query to avoid lock wait times.

Lock mode compatibility for processes that use the SKIP LOCKED DATA option is the same as lock mode compatibility for other page- and row-level locks. However, when incompatible locks are held, a transaction that uses the SKIP LOCKED DATA option does not wait for the locks to be released, but skips the locked data instead. Refer to 2.2.1, “Skip locked data” on page 27 for more information.

### Exception

SKIP LOCKED DATA is ineffective against gross locks. This can be illustrated in the following example scenario.



The mass delete statement acquires table level X lock, which is a gross lock that would be incompatible with the table level IS lock required in the subsequent SELECT (with the SKIP LOCKED DATA option). The SELECT COUNT(\*) will not return zero by skipping all the locked data, but instead it fails with a -911, as shown in Example 5-8.

*Example 5-8 SKIP LOCKED DATA and gross locks*

```
DELETE FROM GLWTEPA

NAME      TYPE PART  STATUS      CONNID  CORRID  LOCKINFO
-----
GLWSEPA   TS      RW      AGENT TOKEN 10624  TS0     DB2R3    H-IX,S,C
-
29        TB      AGENT TOKEN 10624  TS0     DB2R3    H-X,T,C
-
***** DISPLAY OF DATABASE PAOLR5 ENDED *****

SELECT COUNT(*)
FROM GLWTEPA
WITH CS
SKIP LOCKED DATA

DSNT408I SQLCODE = -911, ERROR: THE CURRENT UNIT OF WORK HAS BEEN ROLLED BACK
      DUE TO DEADLOCK OR TIMEOUT. REASON 00C9008E, TYPE OF RESOURCE
      00000D01, AND RESOURCE NAME 00000368.00000029
DSNT418I SQLSTATE = 40001 SQLSTATE RETURN CODE
DSNT415I SQLERRP = DSNXRSFN SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD = -190 -100 0 -1 1026420731 536870912 SQL DIAGNOSTIC
      INFORMATION
DSNT416I SQLERRD = X'FFFFFF42' X'FFFFFF9C' X'00000000' X'FFFFFFF'
      X'C2D21005' X'20000000' SQL DIAGNOSTIC INFORMATION
```

### Concurrency versus accuracy

There are three options for improving concurrency at the expense of accuracy of the returned data, as shown in Table 5-8.

*Table 5-8 Concurrency options versus accuracy*

Control mechanism	Return uncommitted data	Return committed data
Isolation UR on SELECT statement	ALL	ALL
Skip uncommitted inserts DSNZPARM (SKIPUNCI) <sup>a</sup>	NONE	ALL except uncommitted INSERT
SKIP LOCKED ROWS on the SELECT statement	NONE	ALL except uncommitted

a. Refer to the IRLM and DB2 DSNZPARM topics in 7.3.13, “SKIPUNCI: SKIP UNCOMMITTED INSERTS” on page 211.

## 5.8 Unit of recovery and unit of work

Unit of recovery (UOR) is a recoverable sequence of operations within a single resource manager, such as an instance of DB2.

Unit of work (UOW) is a recoverable sequence of operations within an application process. At any time, an application process is a single unit of work, but the life of an application process can involve many units of work as a result of commit or rollback operations. In a multisite update operation, a single unit of work can include several units of recovery. Application programmers must think in terms of the scope of the unit of work when they are designing online as well as batch programs.

A logical unit of work in an application is a set of SQL statements that make up a business process. At any point during the logical unit of work, a failure will require the rollback of all previous updates in that logical unit of work in order to keep the database in a state that is consistent with the business rules. At the end of the logical unit of work, the data manipulated by the business process should be in a state that is consistent with the business rules. In DB2, a unit of recovery is the set of UPDATE, DELETE, or INSERT statements that are performed from one point of consistency to another by a single application process. A unit of recovery begins with the first change to the data after the beginning of the job or following the last point of consistency and ends at a commit point.

In order to ensure that the data in a database remains in a state that is consistent with the business rules, the logical unit of work should be no larger than the DB2 unit of recovery. However, in some cases, the logical unit of work cannot be completed within a single DB2 unit of recovery. An example would be a process that requires three online transactions to complete a business function that is considered a single logical unit of work. Because each online transaction ends with a commit, the rows that have been added or altered in the first transaction are exposed to other processes without being locked. This may also be the case when multiple batch job steps or even multiple batch jobs are required to complete a logical unit of work. In either of these cases, there is also the dilemma of how to handle an abend in a subsequent unit of recovery after previous units of recovery have already been committed.

For best performance, it may be advisable to group multiple logical units of work into a single DB2 unit of recovery. When this is done, savepoints (refer to 5.8.1, "Savepoint" on page 121 for more information about this topic) can be used to allow the application to roll back a single logical unit of work.

Figure 5-6 shows the relationship between a DB2 unit of recovery and a logical unit of work.

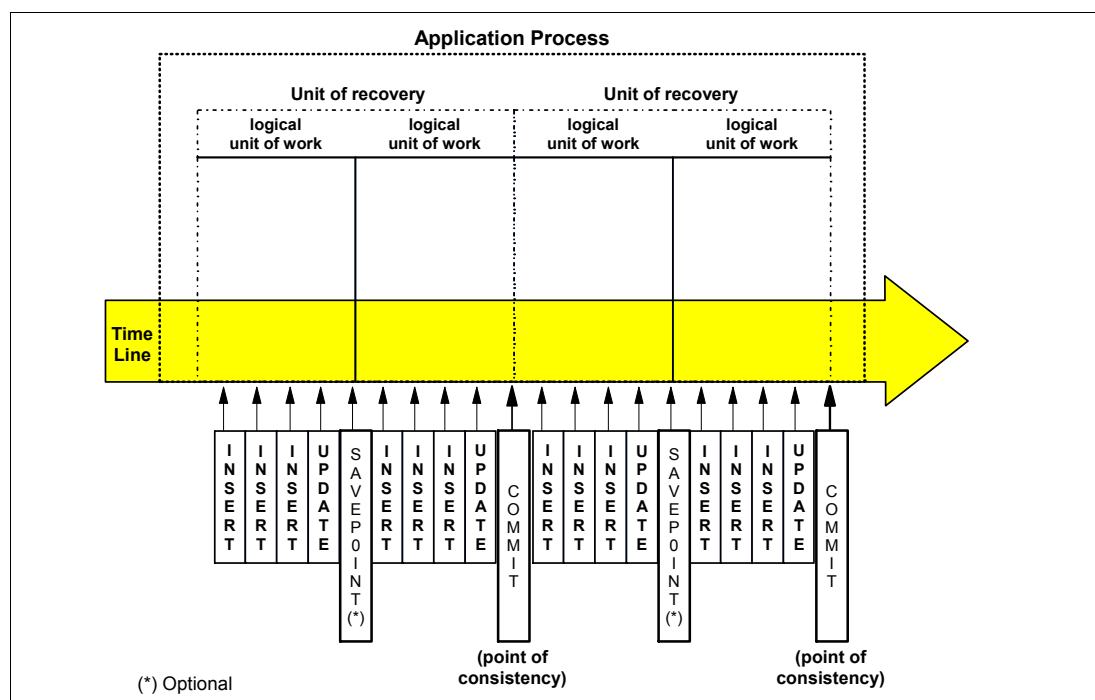


Figure 5-6 Logical units of work and DB2 units of recovery

## 5.8.1 Savepoint

A savepoint represents the state of data at some point in time during a unit of work. Savepoints are set by the application through the use of the SAVEPOINT statement. If subsequent application logic dictates that only a portion of the unit of recovery should be rolled back, the application can perform a rollback to the savepoint.

Savepoints provide applications with a tool to achieve a more granular level of data recoverability without the expense of frequent commits. Issuing a SAVEPOINT command is much less expensive than issuing a COMMIT, because there is no forced write of the log buffer.

Use SAVEPOINT rather than COMMIT to create a point to which your application can roll back for logic purposes. However, be aware that SAVEPOINT or 'ROLLBACK TO SAVEPOINT' do not end a unit of recovery. All locks and claims that are acquired during the unit of recovery are held until commit.

## 5.8.2 Commit

All batch application processes, as well as any other application processes that acquire locks and run for more than a few seconds should perform commits periodically. The following points summarize the reasons for performing commits:

- Any processes that are updating DB2 are acquiring locks on objects that may prevent other processes from accessing these objects. If these locks are held for an unnecessarily long period of time, concurrency will suffer. Excessive long lock duration leads to deadlock and timeout.

- ▶ Any processes that are accessing DB2, including read-only applications, are acquiring claims on objects that may prevent drains from completing. Drains are typically issued by utilities that must be run on the database objects in order to keep them continuously available.
- ▶ In a well designed environment, most read-only applications do not acquire page or row level locks because of lock avoidance. An increase in the number or duration of X locks on pages and rows in the system will reduce the amount of lock avoidance. Acquiring additional locks will add to the elapsed time of read-only transactions and increase the workload in the IRLM address space leading to overall system performance degradation.
- ▶ Whenever a batch program abends or a system failure occurs, DB2 backs out data changes that have not been committed. If no commits are taken, then the amount of data to be backed out may be large and may take a long time. In addition, upon restart the program will have to reapply all of the updates. More frequent commits reduce the amount of data that must be rolled back and then reapplied on restart.

When updating a table space that has a LOCKMAX other than 0 specified, it is important to commit before reaching the maximum number of locks on the table space to avoid lock escalation. You also want to commit before reaching the NUMLKUS threshold, which is the maximum number of locks per user.

Issuing a COMMIT statement ends a unit of recovery and commits all database updates that were made in that unit of recovery. DB2 performs the following functions when a commit occurs:

- ▶ All page and row locks are released, except those held by an open cursor that was declared WITH HOLD.
- ▶ If the RELEASE(COMMIT) bind option was specified, all locks will be released, except for those held by an open cursor that was declared WITH HOLD.
- ▶ All claims are released, except those held by an open cursor that was declared WITH HOLD.
- ▶ If the RELEASE(COMMIT) bind option was specified, the statistics used by the index look-aside and sequential detection processes are discarded.
- ▶ If the RELEASE(COMMIT) bind option was specified, any IPROC or UPROC that had been created by the thread are discarded.
- ▶ A write is forced out of the log buffer to both active logs.

The length of time required for the immediate write of the log buffers depends upon the amount of data that has been updated since the last write of the log buffer. Spreading updates throughout a unit of recovery increases the chances that your log data has been already written before you commit.

**Caution:** While deferring all update, insert, and delete activity until right before you commit can reduce lock duration, it can also increase the amount of log data that must be written at commit, and therefore may cause the commit to take longer.

For more information about the effect of the use of cursors defined WITH HOLD on locking, refer to “SELECT with read-only or ambiguous cursor” on page 90 and “The effect of possible materialization on cursors WITH HOLD” on page 91.

The duration of commit varies. The primary factor is physical log write I/Os. The bind option RELEASE resource at COMMIT causes more impact as more resources need to be freed. As more distinct SQL statements are executed within a commit interval, more resources have to be freed, which takes a longer time. Data sharing commit also includes GBP writes and page P-lock unlocks.

The physical log write I/O of a commit is two synchronous I/O operations for the write of the log buffer to both copies of the active log. In addition to any IPROCs or UPROCs, commit throws away the information used for index look-aside and sequential detection.

DB2 commits updates when a COMMIT is issued either explicitly or implicitly as a result of a termination of a unit of work. The type of DB2 attachment influences when a unit of work is terminated, as shown in Table 5-9.

*Table 5-9 Termination of a unit of recovery*

<b>Attachment</b>	<b>Termination of a unit of recovery</b>
CAF	The program issues a COMMIT/ROLLBACK SQL statement. The program terminates.
RRSAF	The program issues a SRRCMIT/SRRBACK statement. The program terminates.
IMS	The program issues a CKPT, ROLB, or SYNC call. The program issues a GU for the I/O PCB for single-mode transactions. The program terminates.
CICS	The program issues a CICS SYNCPOINT command explicitly. The program terminates as signaled by CICS RETURN. The program issues a DL/I checkpoint or TERM call command.

When designing an application that is going to issue COMMIT statements, be sure to place the COMMIT in the program that controls the logical unit of work. Issuing a COMMIT statement in a sub-program may lead to inadvertently committing in the middle of a logical unit of work. If data access modules are used, the COMMIT statement can be placed in a module that only performs the COMMIT statement, and it should be called only by the program that controls the scope of the logical unit of work.

### 5.8.3 Online processes

There are three basic strategies for online applications performing table updates. Figure 5-7 illustrates the three different strategies for a unit of work.

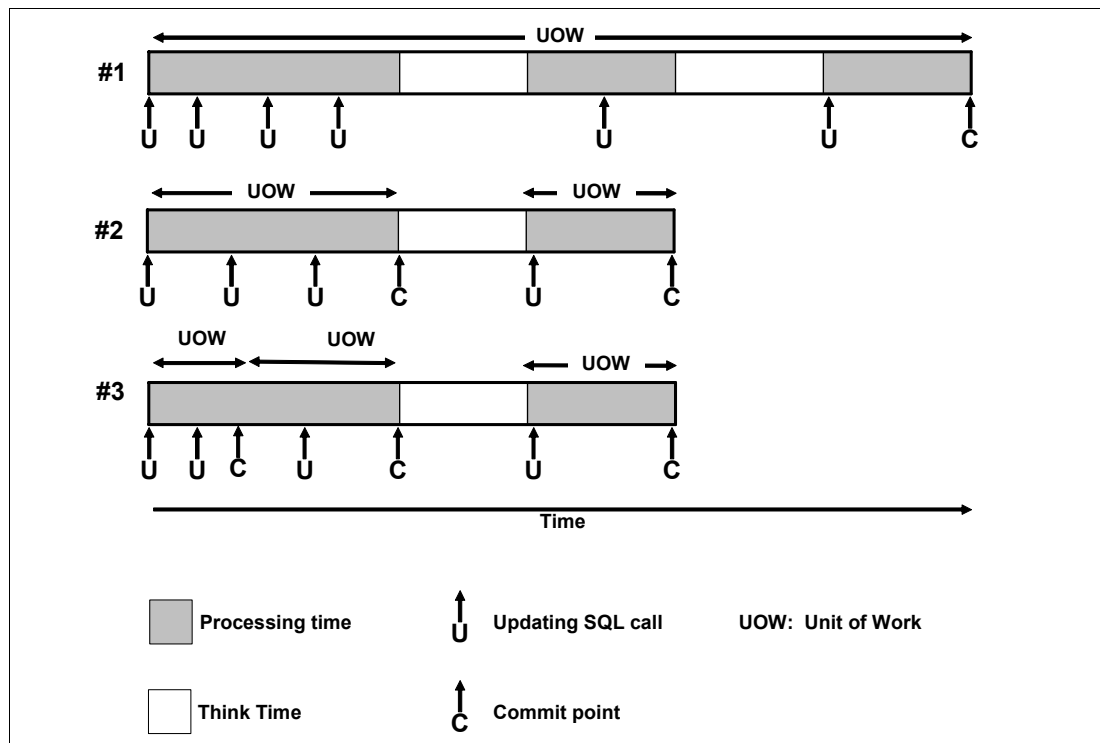


Figure 5-7 DB2 locking strategies for online applications

#### Strategy 1: conversation integrity (not recommended)

A commit point is created at the end of the conversation. The lock duration may include user think time. Therefore, this strategy should not be used in applications where other users need concurrent access to the tables.

Even if there is no need for concurrent processes, this strategy is generally not recommended when you need to update data. DB2 may cancel a program that is inactive (waiting for user input) for more than a few minutes in a distributed environment. This is another reason for always creating a commit point when sending a response to the user.

A read-only application might get away with this strategy in some cases. S locks are normally released before a commit point, but an S-locked page or row cannot be updated by another process and an S lock may have a very long duration. It may include user think time.

We do not recommend this strategy even for read-only applications, although it may be the most convenient one from a programming point of view.

Conversation integrity is the default in TSO, an option in CICS, and not possible in IMS.

## Strategy 2: interaction integrity

A commit point is created (and all locks are released) when the application sends a response to the user.

As no locks are held between user actions, while user A is thinking, user B may update any row. The application program may need logic to detect such situations and to react in a proper way. For example, the application program may have to check row-level timestamps (ROW CHANGE TIMESTAMP), or read the column data again in each transaction to ensure integrity, or specify the WHERE clause of the SQL statement AND COL = :old\_value. It may also be necessary to save the input per user transaction in a private work area (CICS temporary storage or IMS scratch pad area) and perform the actual table updates in the last step.

In online and distributed transactions, the interaction integrity strategy should be the default and, optionally, should include an optimistic concurrency control mechanism.

## Strategy 3: intermediate commit based on clock time

A long-running transaction may take several seconds or minutes. With interaction integrity, such a transaction keeps some pages locked for a long time. If the changes are not strictly related (depending on the business rules), intermediate commit points may be added in the application to reduce waiting time for locks. No commit interval should exceed 5 seconds.

In CICS transactions, an intermediate commit point is created with a SYNCPOINT. In IMS transactions, it is done with a program-to-program switch. In TSO transactions, a COMMIT is used. This strategy should be used for critical Web applications and for batch processes that run concurrently with online/Web applications.

### 5.8.4 Batch processes and commit frequency

Committing frequently to release locks has a cost. The commit frequency must be a compromise between availability and concurrency of data, and also the performance of DB2 applications.

The commit frequency of an application is different depending upon the purpose of the commits. We do not advise setting the commit frequency of all applications to a single value, because the commit frequency that satisfies the requirements of the most highly accessed and critical DB2 objects is probably too frequent for the majority of other processes.

The primary reason for read-only programs to commit is to release the claims that may prevent utilities from draining. The maximum time that a utility will wait for a claim is calculated by multiplying the IRLMRWT DSNZPARM value times the UTIMOUT DSNZPARM value. An appropriate commit frequency for these processes would be half of that time.

For update processes that need to allow concurrent read activity at the row or page level, a general commit frequency would be 25% of the IRLMRWT DSNZPARM value. However, for update processes that access critical, highly accessed rows or pages, a lower percentage might be required.

**Recommendation:** When determining your application's commit frequency, take into consideration the IRLMRWT timeout value and the ability of concurrent processes to wait. A general rule is to commit no more than three times a second, but at least every three to five seconds.

## Manageable units of work

Applications that run for a long time without committing tend to hold resources that may be required by other applications. A failure to code frequent commits, especially for long running batch jobs that run concurrently with online transactions, can cause performance degradation and even deadlocks or timeouts with your transactions. To avoid these circumstances:

- ▶ Ensure that batch jobs include commit logic to release locks held on objects that other applications may require.
- ▶ If your applications access non-DB2 data in addition to DB2 data, then you must design your applications with coordinated sync points to ensure that both DB2 and non-DB2 data can be recovered to the same point in time.

Exploit lock avoidance whenever possible. Lock avoidance is also impacted by the frequency of commit logic. If applications that access the same data concurrently do not commit frequently, you will not be able to take advantage of lock avoidance.

Consider committing using a Checkpoint/Restart framework. Long-running applications, particularly those that perform row-level locking, have few or infrequent commit points, or that use repeatable-read isolation may use substantial amounts of lock storage. You should perform frequent commits to release page/row locks. Long-running UOW should not be allowed, particularly in a data sharing environment.

Consider the potential benefits of LOCK TABLE, if at all applicable.

Not committing frequently is detrimental to concurrency. Committing too frequently would not only result in a poor performance, but could hurt concurrency of all the applications that are trying to do logging. Even though the log write delay is not a major concern, one should keep in mind that a commit is not complete until the log data is externalized to the disk and hence any log write delay will result in an elongated unit of work, resulting in a longer lock duration.

Use LOAD SHRLEVEL CHANGE instead of mass inserts for better concurrency. Use the REORG with DISCARD option with SHRLEVEL CHANGE instead of deleting the records periodically in a batch application.

Lock avoidance is also impacted by the commit frequency. If applications that access the same data concurrently do not commit frequently, your application will not be able to take advantage of lock avoidance.

If the purpose of performing commits in the application is to release page or row locks to allow concurrent batch and online processing, the commit frequency may be selected in the range of 5 seconds for the IRLMRWT (DSNZPARM setting for IRLM timeout) value.

## 5.9 Locks acquired by unconventional SQL statements

In this section, we discuss the locks acquired by the following SQL constructs:

- ▶ MQT REFRESH TABLE statement
- ▶ EXCHANGE DATA BETWEEN clone table and base table
- ▶ TRUNCATE statement
- ▶ Static and dynamic scrollable cursors
- ▶ MERGE statement
- ▶ Data change within a SELECT statement



## 5.9.1 MQT REFRESH TABLE statement

The REFRESH TABLE statement is used to repopulate the Materialized Query Table (MQT) from the current version of the base table. The REFRESH table statement normally acquires an S lock on the base table, but it can be avoided if the MQT was created/alterd with an isolation UR.

The ISOLATION column of the SYSIBM.SYSVIEWS table holds the ISOLATION level at which the MQT was created/alterd. This isolation level of MQT is used for the REFRESH TABLE statement, more precisely, the SELECT portion of the REFRESH statement.

For example, if the MQT was created after setting the ISOLATION level of the create to UR (uncommitted read), then the refresh takes place by reading the base table with ISOLATION UR. This is true for both MAINTAINED BY SYSTEM and MAINTAINED BY USER MQTs. If the ISOLATION level of the MQT is CS (cursor stability) and lock avoidance conditions are satisfied, then the locks acquired by the REFRESH statement on the base table would be same as that for isolation UR. See Example 5-9 and Example 5-10 on page 128 for a sample MQT create statement, along with the locks acquired during REFRESH with isolation UR and RR respectively.

*Example 5-9 Locks acquired by REFRESH statement on an MQT created with ISOLATION(UR)*

```
CREATE TABLE MQT1 AS (SELECT DISTINCT E.EMP_NO, E.FIRSTNME,
    E.LASTNAME, E.WORKDEPT, E.PHONENO, 'RAVI1' AS MANAGER,
    YEAR(S.CREATED_TS) AS YEAR FROM RAVI.GLWTEMP E, RAVI.GLWTDPT S
    WHERE E.WORKDEPT = S.DEPT_NO
    )
    DATA INITIALLY DEFERRED REFRESH DEFERRED MAINTAINED BY USER;

REFRESH TABLE MQT1;
```

Type	Level	Resource		Number
----	-----	-----	-----	-----
DPAG	X	DB=DSNDB06	PS=SYSVIEWS	1
	X	DB=DSNDB06	PS=SYSDBASE	1
DTBS	S	DB=DSNDB07		1
	S	DB=DSN00347		1
	S	DB=RAVI		1
PSET	S	DB=DSNDB07	PS=DSN32K00	1
	IS	DB=DSNDB06	PS=SYSOBJ	1
	IS	DB=DSNDB06	PS=SYSSTATS	1
	IX	DB=DSNDB06	PS=SYSVIEWS	1
	IX	DB=DSNDB06	PS=SYSDBASE	1
PALK	X	DB=DSN00347	PS=MQT1	1
MDEL	S	DB=RAVI	PS=GLWSEMP	1
	S	DB=RAVI	PS=GLWSDPT	1
	X	DB=DSN00347	PS=MQT1	1
	S	DB=DSN00347	PS=MQT1	1
TABL	IS	DB=DSNDB06	PS=SYSOBJ	1
	IS	DB=DSNDB06	PS=SYSSTATS	1

*Example 5-10 Locks acquired by REFRESH statement on an MQT created with ISOLATION(RR)*

```
CREATE TABLE MQT3 AS (SELECT DISTINCT E.EMP_NO, E.FIRSTNME,
E.LASTNAME, E.WORKDEPT, E.PHONENO, 'RAVI1' AS MANAGER,
YEAR(S.CREATED_TS) AS YEAR FROM RAVI.GLWTEMP E, RAVI.GLWTDPT S
WHERE E.WORKDEPT = S.DEPT_NO
)
DATA INITIALLY DEFERRED REFRESH DEFERRED MAINTAINED BY USER;

REFRESH TABLE MQT3;
```

Type	Level	Resource		Number
----	-----	-----	-----	-----
DPAG	S	DB=RAVI	PS=GLWSEMP	274
	X	DB=DSNDB06	PS=SYSVIEWS	1
	X	DB=DSNDB06	PS=SYSDBASE	1
DTBS	S	DB=DSNDB07		1
	S	DB=DSN00349		1
	S	DB=RAVI		1
PSET	S	DB=DSNDB07	PS=DSN32K00	1
	IS	DB=DSNDB06	PS=SYSOBJ	1
	IS	DB=DSNDB06	PS=SYSSTATS	1
	IX	DB=DSNDB06	PS=SYSVIEWS	1
	IX	DB=DSNDB06	PS=SYSDBASE	1
SKCT	S	Plan=DSNESPCS		1
PALK	IS	DB=RAVI	PS=GLWSEMP	4
	S	DB=RAVI	PS=GLWSDPT	1
	X	DB=DSN00349	PS=MQT3	1
MDEL	S	DB=RAVI	PS=GLWSEMP	1
	S	DB=RAVI	PS=GLWSDPT	1
	X	DB=DSN00349	PS=MQT3	1
	S	DB=DSN00349	PS=MQT3	1
TABL	IS	DB=DSNDB06	PS=SYSOBJ	1
	IS	DB=DSNDB06	PS=SYSSTATS	1
IEOF	S	DB=RAVI	PS=GLWXEMP2 PT= 1	1

Upon comparison of the two examples, you will notice that with isolation RR, the base table spaces (GLWSEMP and GLWSDPT) are locked, while there are no page locks or partition locks on the base table with isolation UR. In addition, an index end-of-file lock is also acquired with isolation RR, although mass delete locks are held in both cases on the base tables.

## 5.9.2 EXCHANGE DATA BETWEEN clone table and base table

The EXCHANGE statement switches the content of a base table and its associated clone table. This statement can be embedded in an application program or issued interactively. It is very restrictive.

DB2 has to drive everyone off both the base and the clone objects. Also, because the information in the catalog and directory are being changed, no one can have any access to dependent objects (or allowed to even look at the old information).

Example 5-11 shows the list of locks held by an EXCHANGE statement. When an EXCHANGE statement is running, none of the plans or packages that are dependent on the target table involved can be executed.

*Example 5-11 Locks acquired while an EXCHANGE statement is executed*

Type	Level	Resource		Number
----	-----	-----		-----
DPAG	X	DB=DSNDB06	PS=SYSCOPY	1
	X	DB=DSNDB06	PS=SYSDBASE	1
DTBS	X	DB=DRK02		1
PSET	IX	DB=DSNDB06	PS=SYSCOPY	1
	IX	DB=DSNDB06	PS=SYSPKAGE	1
	IX	DB=DSNDB06	PS=SYSPLAN	1
	IS	DB=DSNDB06	PS=SYSUSER	1
	IX	DB=DSNDB06	PS=SYSXML	1
	IX	DB=DSNDB06	PS=SYSOBJ	1
	IX	DB=DSNDB06	PS=SYSDBASE	1
	IX	DB=DSNDB06	PS=SYSDBAUT	1
SKCT	X	Plan=RAVIPG9B		1
MDEL	X	DB=DRK02		1
	X	DB=DRK02		1
TABL	IX	DB=DSNDB06	PS=SYSPKAGE	1
	IX	DB=DSNDB06	PS=SYSPKAGE	1
	IX	DB=DSNDB06	PS=SYSXML	1
	IX	DB=DSNDB06	PS=SYSOBJ	1
	IX	DB=DSNDB06	PS=SYSOBJ	1

The EXCHANGE statement gets an exclusive lock on the database (DRK02) in which the base table and clone table are defined. Even a DISPLAY DATABASE command would time out and fail if an EXCHANGE statement is not completed within the timeout interval. See Example 5-12 for the resource unavailable message (with timeout as the reason code and database as the resource type).

*Example 5-12 DISPLAY DATABASE command fails when an EXCHANGE statement is running*

DSNT500I	-DB9A DSNTDMST RESOURCE UNAVAILABLE
	REASON 00C9008E
	TYPE 00000100
	NAME DRK02
DSN9023I	-DB9A DSNTDDIS 'DISPLAY DATABASE' ABNORMAL COMPLETION

### 5.9.3 TRUNCATE statement

Lock modes for TRUNCATE depend solely on the type of table space regardless of the LOCKSIZE on table space definition or ISOLATION level. Table 5-10 describes the locking behavior of a TRUNCATE statement for each of the different types of table spaces.

Table 5-10 TRUNCATE statement locking behavior for different table space types

Type of table space	Locks acquired as a result of running a TRUNCATE statement
Universal table space and partitioned table space	There is an X lock on all partitions and a mass delete lock (in X mode) on the table space.
Segmented table space (non-partitioned)	There is an X lock on the table and an IX lock on the table space and a mass delete lock (in X mode) on the table.
Simple table space	There is an X lock on the table space and an mass delete lock (in X mode) on the table space.

Even though TRUNCATE would be a better option than running DELETE without a where clause for partitioned table spaces and simple table spaces (in terms of logging and locking), there are some restrictions that would prevent running a TRUNCATE statement against a table. The TRUNCATE statement would fail with a negative SQLCODE if there are DELETE triggers defined on the table or if the table is a parent in a referential integrity constraint, as shown in Example 5-13.

Example 5-13 Usage restrictions for TRUNCATE table statement

```
DSNT408I  SQLCODE = -20356, ERROR:  THE TABLE WITH DBID=753 AND OBID=3 CANNOT
        BE TRUNCATED BECAUSE DELETE TRIGGERS EXIST FOR THE TABLE, OR THE
        TABLE IS THE PARENT TABLE IN A REFERENTIAL CONSTRAINT
DSNT418I  SQLSTATE  = 428GJ SQLSTATE RETURN CODE
DSNT415I  SQLERRP   = DSNXRSDL SQL PROCEDURE DETECTING ERROR
DSNT416I  SQLERRD   = -230 0 0 -1 0 0 SQL DIAGNOSTIC INFORMATION
DSNT416I  SQLERRD   = X'FFFFFF1A' X'00000000' X'00000000' X'FFFFFFF'
        X'00000000' X'00000000' SQL DIAGNOSTIC INFORMATION
```

Irrespective of the table space type, if you try to read the table concurrently with isolation UR (dirty read) when TRUNCATE is still not completed, the dirty read will wait for the mass delete lock to be released and after the preset IRLM timeout interval, the dirty reader will get a timeout (SQLCODE=-911).

**Note:** Simple table spaces have been deprecated as of DB2 9, and getting rid of them would save you from potential concurrency issues while running a TRUNCATE statement, when multiple tables exist under the same table space.

### 5.9.4 Static and dynamic scrollable cursors

Scrollable cursors can be static or dynamic. Static and dynamic cursors differ in their ability to detect updates, deletes, and inserts into the result set.

#### Static

A read-only cursor. After the cursor is opened, it does not detect any inserts, deletes, or updates that are made by its application, or by any other application.

## Dynamic

A cursor that is sensitive to all inserts, deletes, and updates to the result set that occur after the cursor is opened. A dynamic cursor can insert into, delete from, or update the result set.

The static scrollable cursors are materialized in a table space in the workfile database. The dynamic scrollable cursors do not need to materialize in the workfile database because the scrolling can occur both ways against the active result set using backward index scanning.

With dynamic scrollable cursors, the most recently fetched row or page from the base table remains locked to maintain position for a positioned UPDATE or DELETE. The lock taken at the first FETCH is held until the next FETCH. The lock taken at the last FETCH is held until COMMIT, ROLLBACK, or the end of transaction, as shown in Table 5-11.

*Table 5-11 Cursor concurrency for each cursor type*

Cursor type	Cursor sensitivity	Cursor concurrency
Non-scrollable (forward-only)	Unspecified	Read-only concurrency
Static Scrollable	Insensitive	Read-only concurrency
Dynamic Scrollable	Sensitive	Lock concurrency

### 5.9.5 MERGE statement

The MERGE statement lets you update and insert many rows in a table from a single statement.

This is an easier and more efficient mechanism for merging data using arrays of input.

For example, prior to DB2 9 for z/OS, merging data (from 100 transactions to a master table) required many separate operations:

- ▶ Insert 100 rows from the transactions (and ignore the failed inserts).
- ▶ For every row that currently exist in the master table, perform a separate update operation (up to 100 times).

In addition, you can use the SELECT FROM MERGE statement to return all the updated rows and inserted rows, including column values that are generated by DB2.

A MERGE statement would reduce the total number of locks needed by your application compared to using separate inserts and updates to accomplish the same.

Example 5-14 shows the locks acquired by MERGE.

*Example 5-14 Locks acquired by a sample MERGE statement*

```

MERGE INTO DB2R1.MERGE AS A
USING (VALUES (7111 , 'LINEXXMRG') )
AS T (COLID, COLDATA)
ON (A.COLID = T.COLID)
WHEN MATCHED THEN UPDATE SET COLDATA = T.COLDATA
WHEN NOT MATCHED THEN INSERT (COLID, COLDATA)
VALUES (T.COLID, T.COLDATA);

```

Type	Level	Resource		Number
----	-----	-----		-----
DPAG	X	DB=MERGE	PS=MERGE	1
PSET	IX	DB=MERGE	PS=MERGE	1
	IS	DB=DSNDB06	PS=SYSUSER	1
	IS	DB=DSNDB06	PS=SYSDBAUT	1
	IS	DB=DSNDB06	PS=SYSDBASE	1
TABL	IX	DB=MERGE	PS=MERGE	1

When there is a concurrent UPDATE on the page where the existence check is being performed by DB2, the MERGE could get suspended or possibly time out, even if it is only trying to INSERT a row.

## 5.9.6 Data change within a SELECT statement

Data change statements within a SELECT statement are:

- ▶ SELECT FROM DELETE
- ▶ SELECT FROM UPDATE
- ▶ SELECT FROM INSERT
- ▶ SELECT FROM MERGE

In some cases, a data change within a SELECT statement is better than executing the data change statement (MERGE, INSERT, UPDATE, or DELETE) and the SELECT statement separately. The number of locks requested will always be less in the data change within a SELECT case compared to two statements executed separately.

INSERT, UPDATE, DELETE, and MERGE statements acquire and hold the row/page X lock for commit duration, so the SELECT statement will not acquire another less restrictive S lock on the same row/page when combined.

Example 5-15 shows the locks acquired by a sample SELECT FROM UPDATE statement.

*Example 5-15 Locks acquired by a sample SELECT FROM UPDATE statement*

```
SELECT EMP_NO, SALARY FROM OLD TABLE
(UPDATE RAVI.GLWTEMP SET SALARY = SALARY *1.2
WHERE EMP_NO =1010);
```

Type	Level	Resource		Number
----	-----	-----		-----
DPAG	X	DB=RAVI	PS=GLWSEMP	1
DTBS	S	DB=DSNDB07		1
PSET	S	DB=DSNDB07	PS=DSN8K00	1
	IS	DB=DSNDB06	PS=SYSOBJ	1
	IS	DB=DSNDB06	PS=SYSDBAUT	1
	IS	DB=DSNDB06	PS=SYSSTATS	1
	IS	DB=DSNDB06	PS=SYSDBASE	1
PALK	IX	DB=RAVI	PS=GLWSEMP	1
MDEL	S	DB=RAVI	PS=GLWSEMP	1
TABL	IS	DB=DSNDB06	PS=SYSOBJ	1
	IS	DB=DSNDB06	PS=SYSSTATS	1

The WITH HOLD option has no effect on an SQL data change statement within a SELECT statement. When a COMMIT is issued, the changes caused by the SQL data change statement are committed, regardless of whether or not the cursor is declared WITH HOLD.

Data change with a SELECT statement can also provide some relief in the number of trips across the network in certain types of applications. It is mostly a usability feature. In terms of overall performance, the benefit depends on the work file usage and the work done to execute the two SQL statements together versus the two executed independently.

## 5.10 Concurrency scenarios

A few uncommon locking and concurrency scenarios are discussed in this section in an effort to explain the interaction between different incompatible SQL processes. Using the tables in 5.1, “SQL design” on page 88, the following scenarios can be verified.

### 5.10.1 Scenario 1: lock interaction between concurrent SQL statements

There are four concurrently running processes accessing the same table (T) in a segmented table space with row level locking. The processes are listed in chronological order in Table 5-12.

*Table 5-12 Sample scenario showing lock interaction between SQL statements*

Process	SQL statement	Table lock	Row lock	Lock status
P1	Update row#1	IX	X	Acquired
P2	Update row#2	IX	X	Acquired
P3	Select row#1	IS	S	Waits for P1 (row lock)
P4	LOCK TABLE T IN SHARE MODE	S	None	Waits for P1 and P2 (table lock)

Process P1 executes a searched update statement to change data in row#1 and it requests table lock in IX mode and row lock in X mode (refer to Table 5-1 on page 89). This being the first SQL statement in our sample scenario, the request locks are acquired immediately, as shown in the Lock status column of Table 5-12 on page 133.

Then process P2 executes another searched update statement affecting the data in row#2 only. This would result in the successful acquisition of table lock in mode IX and row lock in mode X (refer to Table 5-1 on page 89), as the affected rows are different and table level IX lock is compatible with another IX lock.

Immediately following process P2 and before releasing the locks acquired in P1 and P2, another concurrent process P3 starts and executes a SELECT statement (this could be with any isolation level except uncommitted read). This requests table lock in mode IS and row lock in mode S (refer to Table 5-1 on page 89). Because this process is incompatible with P1 (refer to the compatibility matrix in Table 2-1 on page 18), this would result in a lock suspension. Thus, P3 would wait until P1 to release the X lock on row#1 and may eventually time out.

Subsequently, if another process, P4, starts to execute a LOCK TABLE statement in SHARE MODE, then this would send a gross lock request to IRLM with a lock size of table in share mode. Because the table level S lock is not compatible with IX lock (refer to Table 2-2 on page 19) being held by P1 and P2, P4 will enter into a wait state. If the IX locks (held by P1 and P2) are not released within the IRLMRWT timeout interval since the time at which the lock is requested by P4, it will get a timeout error.

If the X and IX locks are released quick enough to avoid timeouts, then the waiting applications would only experience lock wait, which may affect the elapsed time, but will not result in any disruption by way of timeouts.

## 5.10.2 Scenario 2: INSERT timeouts

If there is a gross lock on the object either through lock escalation or through a LOCK TABLE statement, then the inserts will wait and time out eventually. This could happen even if there are no referential integrity constraints defined on this table. This scenario is shown in Table 5-13.

Table 5-13 Sample scenario showing a INSERT timeout

Process	Short description of the nature of SQL statements	Table lock	Page lock	Lock status
P1	SELECT with ISO(CS) CD(NO)	IS	Avoided	Table lock acquired
P2	DELETE NUMLKTS+1 pages worth of data from T (through a single searched delete statement)	IX (originally requested) to X (newly requested)	X	Waits for P1 (table level lock)
P3	INSERT into T	IX	X	Waits for P2 (when lock escalation is in progress)

There are three concurrently running processes accessing the same table (T) defined in a segmented table space with page level locking. The processes are listed in chronological order.



Process P1 is a long running batch application that reads table T by way of a CURSOR by employing a lock avoidance technique. No page locks are requested during the FETCH, but a table lock is successfully acquired in IS mode.

Process P2, which runs concurrently, tries to delete several rows of data using one searched delete statement. When the number of pages touched exceeds the NUMLKTS setting, DB2 decides to escalate the page level lock to table level gross lock in X mode and sends a change request to IRLM. However, the escalation request will not be granted, as an incompatible lock is being held by process P1. Process P2 waits for P1 to complete.

Process P3 now tries to INSERT into T and requests an IX lock at the table level, but this IX request waits for P2 to promote its IX lock to X, and if P1 is not releasing the IS lock, P2 will not be able to complete lock escalation, so P3 receives a timeout after the IRLMRWT timeout interval. If P1 releases the IS lock on the table, P2 will be able to complete lock escalation successfully, but even then the INSERT would fail if the X lock acquired by P2 is not released within the timeout interval.

In this scenario, these three seemingly compatible processes turned out to be incompatible because of the lock escalation. When the three concurrently running processes are incompatible, the second process requests a lock on the resource that is currently incompatible with the target lock state of the first process, and the third process waits for the second process to acquire and release the lock on this resource. In short, the locks are acquired on a “first come, first served” basis for a given resource.

This concurrency issue could have been avoided if there had been no lock escalation (that is, if less number of pages are locked or if the NUMLKTS setting is higher) or if isolation UR had been used on the long running read process (P1).

There will not be any conflict ever at the page level during INSERT, which normally requests a conditional lock. Refer to “Conditional versus unconditional” on page 45 for more information about conditional locks.

## 5.11 LOB and XML locks

For XML and LOBs, DB2 uses locks to ensure that an application cannot access partial or incomplete data.

### 5.11.1 LOB locks

The purpose of a LOB lock is different than that of regular transaction locks. A lock that takes on a LOB value in a LOB table space is called a LOB lock. LOB column values are stored in a different table space, called a LOB table space, from the values in the base table. An application that reads or updates a row in a table that contains LOB columns obtains its normal transaction locks on the base table. The locks on the base table also control concurrency for the LOB table space. When locks are not acquired on the base table, such as for ISO(UR), DB2 maintains data consistency by using LOB locks.

#### ISOLATION (UR)

When an application is reading rows using uncommitted read, no page or row locks are taken on the base table. Therefore, these readers must take an S LOB lock to ensure that they are not reading a partial LOB or a LOB value that is inconsistent with the base row. This LOB lock is acquired and released immediately, which is sufficient for DB2 to ensure that a complete copy of the LOB data is ready for subsequent reference.

## Hierarchy of LOB locks

Just as page, row, and table space locks have a hierarchical relationship, LOB locks and locks on LOB table spaces have a hierarchical relationship. If the LOB table space is locked with a gross lock, then LOB locks are not acquired. In a data sharing environment, the lock on the LOB table space is used to determine whether the lock on the LOB must be propagated beyond the local IRLM.

## Modes of LOB locks

The following LOB lock modes are possible:

- ▶ S (SHARE)

The lock owner and any concurrent processes can read, update, or delete the locked LOB. Concurrent processes can acquire an S lock on the LOB.

- ▶ X (EXCLUSIVE)

The lock owner can read or change the locked LOB. Concurrent processes cannot access the LOB.

The following lock modes are possible on the LOB table space:

- ▶ IS (INTENT SHARE)

The lock owner can update LOBs to null or zero-length, or read or delete LOBs in the LOB table space. Concurrent processes can both read and change LOBs in the same table space.

- ▶ IX (INTENT EXCLUSIVE)

The lock owner and concurrent processes can read and change data in the LOB table space. The lock owner acquires a LOB lock on any data it accesses.

- ▶ S (SHARE)

The lock owner and any concurrent processes can read and delete LOBs in the LOB table space. An S lock is only acquired on a LOB in the case of an ISO(UR).

- ▶ SIX (SHARE with INTENT EXCLUSIVE)

The lock owner can read and change data in the LOB table space. If the lock owner is inserting (INSERT or UPDATE), the lock owner obtains a LOB lock. Concurrent processes can read or delete data in the LOB table space (or update to a null or zero-length LOB).

- ▶ X (EXCLUSIVE)

The lock owner can read or change LOBs in the LOB table space. The lock owner does not need LOB locks. Concurrent processes cannot access the data.

## LOB lock duration

Locks on LOBs are taken when they are needed for an INSERT or UPDATE operations and released immediately at the completion of the operation. LOB locks are not held for SELECT and DELETE operations. In the case of an application that uses the uncommitted read option, a LOB lock might be acquired, but only to test the LOB for completeness. The lock is released immediately after it is acquired.

Locks on LOB table spaces are acquired when they are needed, that is, the ACQUIRE option of BIND has no effect on when the table space lock on the LOB table space is taken. When the table space lock is released is determined by a combination of factors:

- ▶ The RELEASE option of bind
- ▶ Whether the SQL statement is static or dynamic
- ▶ Whether there are held cursors or held locators

When the release option is COMMIT, the lock is released at the next commit point, unless there are held cursors or held locators. If the release option is DEALLOCATE, the lock is released when the object is deallocated (the application ends). The BIND option has no effect on dynamic SQL statements, which always use RELEASE(COMMIT), unless you use dynamic statement caching.

The LOB table space associated with a LOB column is not locked if the application:

- ▶ Selects a LOB that is null or zero length
- ▶ Deletes a row where the LOB is null or zero length
- ▶ Inserts a null or zero length LOB
- ▶ Updates a null or zero-length LOB to null or zero-length

LOB locks are counted toward the total number of locks allowed per user. Control this number with the value you specify in the LOCKS PER USER field of installation window DSNTIPJ. As with any table space, the LOCKMAX clause of the CREATE TABLESPACE or ALTER TABLESPACE statement is used to control the number of LOB locks that are acquired within a particular LOB table space. Lock counts are always kept on a table space level. For an application process that is accessing LOBs, the LOB lock count on the LOB table space is maintained separately from the base table space, and lock escalation occurs separately from the base table space.

### Explicitly locking LOB tables

The reasons for using LOCK TABLE on an auxiliary table are somewhat different than that for regular tables:

- ▶ LOCK TABLE can be used to control the number of locks acquired on the auxiliary table and to eliminate the need for lower-level LOB locks.
- ▶ LOCK TABLE IN SHARE MODE can be used to prevent other applications from inserting LOBs. With auxiliary tables, LOCK TABLE IN SHARE MODE does not prevent any changes to the auxiliary table. The statement does prevent LOBs from being inserted into the auxiliary table, but it does not prevent deletes. Updates are generally restricted also, except where the LOB is updated to a null value or a zero-length string.
- ▶ LOCK TABLE IN EXCLUSIVE MODE can be used to prevent other applications from accessing LOBs. With auxiliary tables, LOCK TABLE IN EXCLUSIVE MODE also prevents access from uncommitted readers.

### Controlling lock size for LOB table spaces

The LOCKSIZE TABLE, PAGE, ROW, and XML options are not valid for LOB table spaces. The other options act as follows:

- ▶ LOCKSIZE TABLESPACE: A process acquires no LOB locks.
- ▶ LOCKSIZE ANY: DB2 chooses the size of the lock. For a LOB table space, this is usually LOCKSIZE LOB.
- ▶ LOCKSIZE LOB: If LOBs are accessed, a process acquires the necessary LOB table space locks (IS or IX), and might acquire LOB locks.

## 5.11.2 XML locks

DB2 uses a locking scheme for concurrency on XML data to achieve document level consistency and row level consistency. To achieve document level consistency, an XML lock was introduced. XML lock is a lock on an XML document. The same old lock modes and compatibility matrix are used for XML locks. XML locks are used to prevent uncommitted readers from reading an incomplete document being inserted or updated. They are also used to protect XML document references materialized into workfiles during query processing to provide row consistency. That is, a lock is held if an XML column reference is saved into a workfile together with other base table columns.

Table 5-14 shows the XML locks for various SQL operations.

*Table 5-14 XML locks for various types of SQL statements*

SQL	Base page/row lock (Business as usual)	XML lock (Size=XML document)	XML table space page lock
INSERT	X page/row lock	X lock, release at commit	Page latch and optional P-lock
UPDATE/DELETE	U->x, S->X, X stays x	X lock, release at commit	Page latch and optional P-lock
SELECT UR, CS-CDN	None	S lock, release at next row fetch	None
SELECT CS-CDY no workfile	S page/row lock, release on next row fetch	S lock, release at next row fetch	None
SELECT UR, CS-CDN, CS-CDY with multirow fetch and dynamic scrolling	S page/row lock on rowset, release on next fetch	S lock, release on next fetch	None
SELECT RR, RS	S page/row lock	S lock, release at commit	None

## 5.12 Access path influence on locking

The access path that DB2 uses can affect the mode, size, and even the object of a lock.

For example, an UPDATE statement using a table space scan might need an X lock on the entire table space. If rows to be updated are located through an index, the same statement might need only an IX lock on the table space and X locks on individual pages or rows.

If you use the EXPLAIN statement to investigate the access path chosen for an SQL statement, then check the lock mode in column TSLOCKMODE of the resulting PLAN\_TABLE. If the table resides in a non segmented table space, or is defined with LOCKSIZE TABLESPACE, the mode shown is that of the table space lock. Otherwise, the mode is that of the table lock.

### 5.12.1 Index-only access and data-only locking

No index page locks are acquired during processing. Instead, DB2 uses a technique called data-only locking to serialize changes. There are no explicit index space locks. However, all index spaces on tables are implicitly locked at the same level as the table spaces containing those tables. In a data sharing environment, P-locks are acquired by DB2 on index spaces as well as on table spaces.

Index page latches are acquired to serialize changes within a page and keep the page physically consistent. Acquiring page latches ensures that transactions accessing the same index page concurrently do not see the page in a partially changed state.

The underlying data page or row locks are acquired to serialize the reading and updating of index entries to ensure the data is logically consistent, meaning that the data is committed and not subject to rollback or abort. The data locks can be held for a long duration, such as until commit. However, the page latches are only held for a short duration while the transaction is accessing the page. Because the index pages are not locked, hot spot insert scenarios (which involve several transactions trying to insert different entries into the same index page at the same time) do not cause contention problems in the index.

A query that uses index-only access might lock the data page or row, and that lock can contend with other processes that lock the data. However, using lock avoidance techniques can reduce the contention. Refer to 2.3, “Lock avoidance” on page 33 for more information about how lock avoidance works when the access path chosen was index-only.

Figure 5-8 shows how DB2 handles locking when index-only access is used by the optimizer. When ISOLATION(CS) is specified for a read-only or ambiguous cursor, DB2 acquires an S lock at the row or page level depending on the LOCKSIZE of the table space. The index-only access path for a given query is indicated by INDEXONLY=Y on the PLAN\_TABLE. The left side of Figure 5-8 shows a data-only locking scenario for LOCKSIZE=PAGE; the right side shows a scenario for LOCKSIZE=ROW.

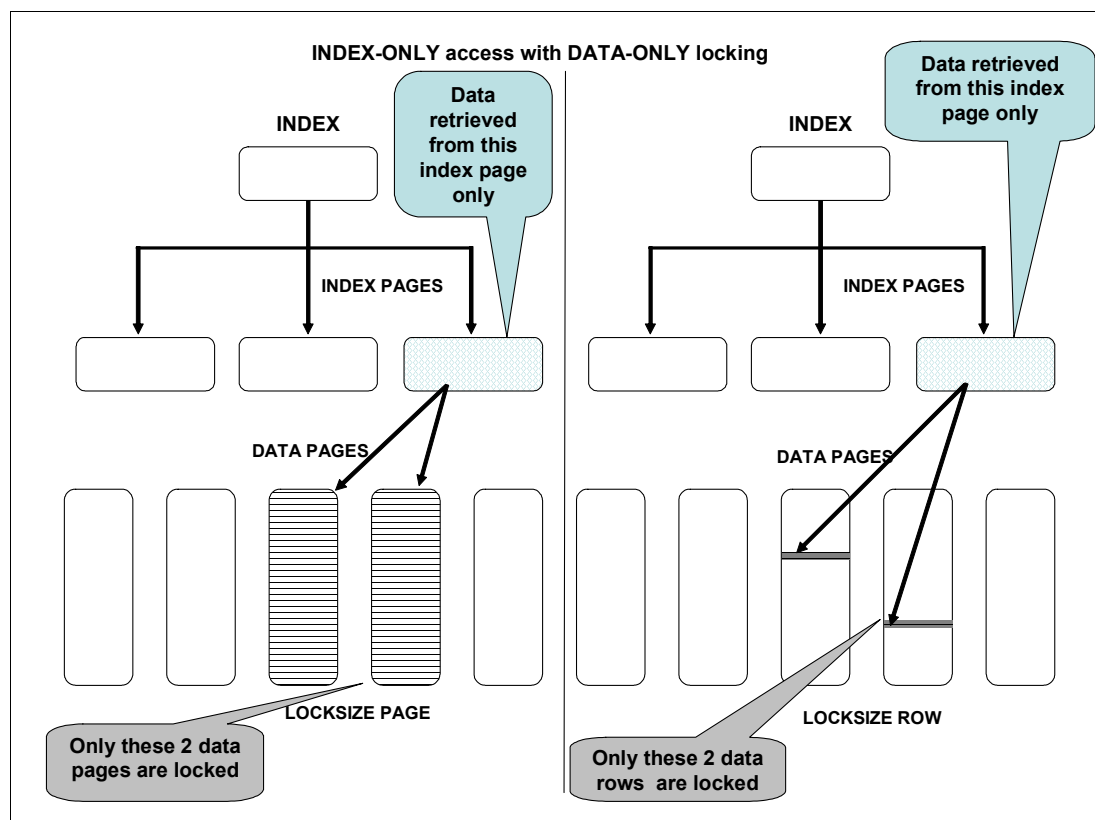


Figure 5-8 Data-only locking for index-only access with lock size row/page

To provide better concurrency, when DB2 searches using XML values (the first key values) in the XML index key entries, it does not acquire the index page latch and does not lock either the base table data pages or rows, or the XML table space.

When the matched-value index key entries are found, the corresponding DOCID values (the second key value) are retrieved. The retrieved DOCID values are used to retrieve the base table RIDs using the DOCID index. At this time, the regular data-only locking technique is applied on the DOCID index page and base table data page (or row).

## 5.12.2 Temporary result tables and lock avoidance

Temporary result tables use a workfile database. Depending on the access path, DB2 may build a result table when a read-only cursor is opened. In that case, acquiring and releasing of locks takes place with OPEN CURSOR while the temporary table is being built. When the OPEN CURSOR processing is completed, no locks remain if CS isolation is used. Rows are retrieved from the temporary table and no locking is done at FETCH time. This is true irrespective of the value of CURRENTDATA.

**Note:** When temporary result tables are built by DB2, CURRENTDATA(YES) behavior is similar to CURRENTDATA(NO) behavior from the application point of view.

Lock duration is shorter than when the result table is built row by row at FETCH time.

Dynamic scrollable cursors do not use the workfile database. The static scrollable cursors are materialized in a table space in the workfile database.

### 5.12.3 Table space scan with repeatable reads

With repeatable read, implicit lock promotion occurs for table space scan to prevent the insertion of rows that might qualify for the predicate. If access is by way of index, DB2 locks only the key value. If access is by way of table space scans, DB2 locks the table, partition, or table space. This lock promotion happens even if the table space was defined with LOCKSIZE=PAGE or ROW or ANY.

### 5.12.4 EXPLAIN statement

The EXPLAIN statement, or the EXPLAIN option of the BIND and REBIND subcommands, can be used to determine which modes of table, partition, and table space locks DB2 initially assigns for an SQL statement.

TSLOCKMODE: EXPLAIN stores its results in a table called PLAN\_TABLE. To review the results, query the PLAN\_TABLE. After running EXPLAIN, each row of PLAN\_TABLE describes the processing for a single table, either one named explicitly in the SQL statement that is being explained or an intermediate table that DB2 has to create. The column TSLOCKMODE of PLAN\_TABLE shows an initial lock mode for that table. The lock mode applies to the table or the table space, depending on the value of LOCKSIZE and whether the table space is segmented or non segmented.

Table 5-15 on page 142 can be used in conjunction with the TSLOCKMODE column to figure out what table or table space lock is used and whether page or row locks are used also, for the particular combination of lock mode and LOCKSIZE in which you are interested.

If the access method in the ACESSTYPE column is DX, DI, or DU, no latches are acquired on the XML index page and no lock is acquired on the new base table data page or row, nor on the XML table and the corresponding table spaces. The value of TSLOCKMODE is a blank in this case.

EXPLAIN gathers information only about data access in the subsystem where the EXPLAIN statement is run or the bind operation is carried out. To analyze the locks obtained at a remote DB2 location, you must run EXPLAIN at that location.

**Note:** Lock modes for DB2 referential integrity and for LOB and XML table spaces are not reported with EXPLAIN.

The five most commonly occurring TSLOCKMODE values are IX, IS, X, U, and S. When TSLOCKMODE = N, it implies UR isolation and no lock for normal operation (though there will be a mass delete lock at a higher level in S mode).

Table 5-15 can be used to interpret the TSLOCKMODE values on the PLAN\_TABLE for various types of table spaces and LOCKSIZES.

Table 5-15 How to interpret TSLOCKMODE for various table space types and LOCKSIZES

Table space type	Lock level	TSLOCKMODE from PLAN_TABLE				
		IS	S	IX <sup>a</sup>	U	X
For universal table spaces, partitioned table spaces, and simple table spaces (V8) with LOCKSIZE ANY, ROW, or PAGE	Table space lock requested	IS	S	IX	U	X
	Page or row locks requested	Yes	No	Yes	No	No
For segmented table spaces <sup>b</sup> with LOCKSIZE ANY, ROW, or PAGE	Table space lock requested	IS	IS	IX	N/A	IX
	Table locks requested	IS	S	IX		X
	Page or row locks requested	Yes	No	Yes		No
For segmented table spaces <sup>b</sup> with LOCKSIZE TABLE	Table space lock requested	N/A	IS	N/A	IX	IX
	Table lock requested		S		U	X
	Page or row locks requested		No		No	No
For LOCKSIZE TABLESPACE on any type of table spaces other than segmented table space	Table space lock requested	N/A	S	N/A	U	X
	Page or row locks requested		No		No	No

a. Even though the lock requested is IX, DB2 may end with a SIX lock, if there is a gross S lock already in effect.

b. And simple tables spaces in DB2 9.

If the isolation level cannot be determined at bind time, the lock mode is determined by the isolation level at run time and is shown by the values given in Table 5-16.

Table 5-16 Lock mode as determined by the isolation level at run time

TSLOCKMODE from PLAN_TABLE			
NS	NIS	NSS	SS
For UR isolation, no lock (N)			For UR, CS, or RS isolation, an IS lock
For CS, RS, or RR, an S lock	For CS, RS, or RR, an IS lock	For CS or RS, an IS lock, and for RR, an S lock	For RR isolation, an S lock



## 5.13 CICS: locking considerations

In a CICS environment, concurrency is likely to be high. To give maximum concurrency, you should use row level locking or page locking instead of table or table space locking. NUMLKTS should be set to a value so high that lock escalation does not take place for a normal CICS operation.

In general, we recommend that you design CICS programs so that:

- ▶ Locks are kept as short as possible.
- ▶ The number of concurrent locks is minimized.
- ▶ The access order of the tables is the same for all transactions.
- ▶ The access order of the rows inside a table is the same for all transactions.

### 5.13.1 Protected entry threads

Using RELEASE DEALLOCATE to reduce locking overhead is not the solution to all your locking problems, as it incurs storage and storage management overhead.

If you have a high number of SQL statements (for example, 20 or more), then the savings from thread reuse will be lost to storage management overhead. For example, with more than 20 transactions per second, you may get 1% savings from thread reuse, but the savings will be lost to storage management overhead. In general, if you have less than 10 SQL statements (less than 10 transactions per second), this would yield better performance and better concurrency.

The LOCK TABLE statement should be avoided, unless it is strictly necessary. If the LOCK TABLE statement is used in an online program, it can prevent the use of RELEASE(DEALLOCATE) and of protected threads. If you do use a LOCK TABLE statement, your plan should use the bind option RELEASE(COMMIT).

### 5.13.2 Pseudo-conversation and locking

There is a technique in CICS called pseudo-conversational processing, in which a series of non-conversational transactions appears as a single conversational transaction. No transaction exists while the user waits for input; CICS takes care of reading the input when the user gets around to sending it. Thus, in a conversational transaction, programs hold locks while waiting to receive data. In a pseudo-conversational model, no locks are held during these waits.

## 5.14 Distributed application considerations

In a distributed environment, each server performs their own locking and would not interfere with each other.

**Note:** If you want two independent threads to share locks (use the same lock token), then you would have to set up a global transaction and indicate that both threads/connections of the global transaction want to share resources.

Applications that use repeatable read can leave rows or pages locked for longer periods, especially in a distributed environment, and they can claim more logical partitions than similar applications using cursor stability. They are also subject to being drained more often by utility operations. Because so many locks can be taken, lock escalation might take place. Frequent commits release the locks and can help avoid lock escalation.

The KEEP\_DYNAMIC option has locking implications for DRDA clients that specify WITH HOLD on their cursors.

If KEEP\_DYNAMIC(YES) is specified, the DB2 for z/OS server automatically closes the cursor when SQLCODE +100 is detected, which means that the client does not have to send a separate message to close the held cursor. Apart from reducing network traffic, it also reduces the duration of locks that are associated with the held cursor.

A little-known effect of RELEASE(DEALLOCATE) that applies to all packages, whether they contain static or dynamic SQL, regardless of the KEEP\_DYNAMIC option, is that the package itself is kept by the thread at commit, that is, the copy of the PT (package table) in the EDM pool, owned by this thread, is not released. That means that in the first SQL in the next unit of work, the thread has to go back to EDM to get a new copy. This is normally not a problem, because the corresponding SKPT will almost always be in the EDM pool already. It can, however, become a performance problem for an application using ODBC or JDBC that specify autocommit=yes, thereby committing after every statement.

## Stored procedures

Locks on DB2 tables are not held across network transmissions for stored procedures, thereby reducing contention for resources at the server. If the timeout or deadlock occurs inside the stored procedure, then DB2 will not attempt to do rollback, even when called from a program using TSO (or CAF) attach. You are not considered to be “inside the stored procedure:” when fetching from the result set.

### *Using the COMMIT ON RETURN YES clause*

Consider using the COMMIT ON RETURN YES clause of the CREATE PROCEDURE statement to indicate that DB2 should issue an implicit COMMIT on behalf of the stored procedure upon return from the CALL statement. Using the clause can reduce the length of time locks are held and can reduce network traffic. With COMMIT ON RETURN YES, any updates made by the client before calling the stored procedure are committed with the stored procedure changes.

## Cursor WITH HOLD

If you define a cursor using the WITH HOLD option, the locks it needs can be held past a commit point.

Use the CLOSE CURSOR statement as soon as possible in your application to cause those locks to be released and the resources that they hold to be freed at the first commit point that follows the CLOSE CURSOR statement. CLOSE CURSOR is needed to allow the distributed thread to become inactive so that it can be reused.

Do not hold on to cursors you do not need. Avoid CURSOR WITH HOLD (which is the default) at all costs. Turn it off by setting CURSORHOLD=NO.

### **LOCK TABLE statement**

LOCK TABLE has no effect on locks that are acquired at a remote server, that is, the LOCK TABLE statement affects the table only for the currently connected database server. The LOCK TABLE statement is not propagated to all database servers that have participated in the transaction, but if the threads are part of the same global transaction and have indicated that they want to share resources, then those threads will not contend with each other for those threads.

### **Pooled database access thread**

An idle thread that is waiting for a new unit of work from a connection to another system so that it can begin is called a pooled DBAT. Pooled threads hold no database locks.

### **Deadlock detection when multiple DB2 subsystems are involved**

It is possible for two processes to be running on two different distributed DB2 subsystems, each trying to access a resource at the other location. In that case, neither subsystem can detect that the two processes are in deadlock; the situation resolves only when one process times out.

## **5.15 Dynamic SQL considerations**

Here we discuss few aspects of concurrency that are directly related to dynamic SQL.

### **5.15.1 DBD locks**

We have seen in 2.1.5, “Dynamic statement concurrency” on page 23 that, without statement caching, dynamic DML SQL statements acquire an S lock on the DBD at PREPARE time and hold it until the thread reaches a commit point. When global statement caching is specified and active with (CACHEDYN=YES), no DBD lock is acquired.

### **5.15.2 Bind option RELEASE(DEALLOCATE)**

The bind options ACQUIRE and RELEASE determine when DB2 acquires and releases locks a table space, partition, or a table that your application uses. These locks are sometimes called gross or parent locks. The options apply only to static SQL statements, which are bound before your program executes. They do not affect dynamic SQL statements. Their locks are released at commit.

But there is one exception. On subsystems where full caching is used (combining local and global dynamic statement caching), RELEASE(DEALLOCATE) also applies for dynamic SQL. When full caching is active, DB2 retains dynamically prepared SELECT, INSERT, UPDATE, and DELETE statements in memory past commit points. For this reason, DB2 can honor the RELEASE(DEALLOCATE) option for these dynamic statements. The gross locks are held until deallocation, or until the commit after the prepared statement is freed from memory. Refer to Chapter 8. “Dynamic caching”, in *DB2 for z/OS and OS/390 : Squeezing the Most Out of Dynamic SQL*, SG24-6418 for more information. When using RELEASE(DEALLOCATE) in a full caching environment, DB2 will still try to demote the gross lock at commit time, if possible. If an S lock, SIX lock, or X lock is held, DB2 tries to demote it to an IS lock or IX lock, if possible.

With RELEASE(COMMIT), the use count of the DBDs referenced by the application are decreased by one. DBDs are “stealable” if their use count is zero. With RELEASE(DEALLOCATE), the use count is only decreased when the thread terminates. Therefore, it takes significantly longer before the use count of a DBD reaches zero and becomes “stealable”. Otherwise, a DBD is only “stealable” if, for all statements that referenced the DBD, the statement name is reused by statements that do not use this DBD. If DBDs are kept longer, they may take up storage in the EDM pool.

## 5.16 Programming for the Instrumentation Facility Interface

The Instrumentation Facility Interface (IFI) is designed for programs that need online trace information. IFI can be accessed through any of the DB2 attachment facilities.

### 5.16.1 Locking considerations for IFI

When you design your application to use IFI, consider the potential for locking delays, deadlocks, and timeout conflicts.

Locks are obtained for IFI in the following situations:

- ▶ When READS and READA<sup>2</sup> requests are checked for authorization, short duration locks on the DB2 catalog are obtained. When the check is made, subsequent READS or READA requests are not checked for authorization. Remember, if you are using the access control exit routine, then that routine might be controlling the privileges that the monitor trace can use.
- ▶ When DB2 commands are submitted, each command is checked for authorization. DB2 database commands obtain additional locks on DB2 objects.

A program can issue SQL statements through an attachment facility and DB2 commands through IFI. This environment creates the potential for an application to deadlock or time out with itself over DB2 locks acquired during the execution of SQL statements and DB2 database commands. You should ensure that all DB2 locks acquired by preceding SQL statements are no longer held when the DB2 database command is issued. You can do this by:

- ▶ Binding the DB2 plan with ACQUIRE(USE) and RELEASE(COMMIT) bind parameters
- ▶ Initiating a commit or rollback to free any locks your application is holding, before issuing the DB2 command

---

<sup>2</sup> The READS and READA functions allow a monitor program to, respectively synchronously or asynchronously, read trace data that has accumulated in an OPn buffer.

## 5.17 Data integrity exposure issues

Without an effective means of concurrency control, the data integrity could be compromised. The following three scenarios summarize the common issues of data integrity exposure that were introduced in 1.3, “The reasons for serialization” on page 5:

- ▶ **Lost updates**

Suppose that process A and process B both read the same row from the database, and both calculate a new value for one of its columns. If process A updates the row with its new value first, and subsequently process B then updates the same column in the same row, the first update (by process A) is lost. This condition should be prevented by choosing the proper isolation level or appropriate application design.

- ▶ **Unrepeatable reads**

Certain application processes may require the following sequence of events: process A reads a particular row from a table and then proceeds to other SQL requests. At a later time, process A reads the first row again and must find the same values in all columns as when it read that row the first time. Lacking proper concurrency control, another application could have changed the row between the two read operations. This condition should be prevented by using appropriate isolation levels (either RR or RS) or locking the table for the duration of the application process, if needed.

- ▶ **Access to uncommitted data**

Consider the following scenario: Process A updates some column values in a row, and before that change is committed, process B reads in the new (update) values for that row. If process A then issues a rollback (either by an explicit ROLLBACK statement in the program logic, or an automatic rollback by DB2 because of an error condition), then process B has a row that is based upon uncommitted (and therefore potentially incorrect) data. Even though DB2 allows this through isolation level UR (uncommitted read), it should be used carefully. In a pseudo-conversational environment and in some decision support systems where reading of dirty data is not a cause for concern, it may be prudent to access uncommitted data without causing any data integrity issues.

## 5.18 Performance considerations

Locking performance is affected by a number of factors, such as:

- ▶ **Bind parameters**

Parameters, such as ACQUIRE, RELEASE, ISOLATION, and CURRENTDATA, affect how and when locks are taken for a specific plan or package.

- ▶ **Table space parameters**

Parameters, such as LOCKSIZE and LOCKMAX, control the method of locking that is used for each table space. The MAXROWS parameter can be used to influence the granularity of locking on a table space by reducing the number of rows on a page.

- ▶ **IRLM subsystem parameters**

Parameters at the IRLM level control the amount of time that DB2 threads will wait for locked resources and the frequency with which they will be checked for deadlock and timeout conditions.

- ▶ **DB2 subsystem parameters**

DSNZPARMs, such as NUMLKTS, NUMLKUS, EVALUNC, RRULOCK, SKIPUNCI, and XLKUPDLT, control locking behavior at a DB2 subsystem level.

- SQL coding

The use of SQL clauses, such as WITH HOLD, WITH isolation-level, FOR UPDATE OF, and FOR FETCH ONLY, communicate to DB2 the locking requirements of a specific SQL statement.

Choosing multirow fetch and multirow insert can reduce locking and improve performance.

- The order of SQL statement execution

The order of execution of SQL statements and their placement within the logical unit of work can affect the duration of locks.

- The frequency of commits

For DB2 processes that perform more than one logical unit of work, the frequency of commits has a significant effect on the locking behavior of the process and all other processes accessing the same DB2 objects concurrently.

All these factors are discussed in detail in this book.

Similarly, poorly performing SQLs have the tendency to lengthen the lock duration, so tuning your SQL and your database environment would help improve the concurrency by reducing the duration of locks held. For example, consider the following factors:

- When an application commits, DB2 has to wait till the log write operation is complete before releasing the qualifying locks, so improving the log write performance would benefit locking indirectly.
- Regarding the buffer pool tuning, less synchronous reads on a randomly accessed page is good.
- The CPU queuing time should be carefully studied and eliminated.

## Comparison of LOCKSIZE and CURRENTDATA setting

In order to get some idea of the consequences of the choice of the different options, we used the Workload Generator tool to execute an identical workload with different choices for the LOCKSIZE. For the data sharing runs, the workload was split in half and run on two members of a data sharing group. Inevitably, this workload cannot be guaranteed to be identical. The results are shown in Table 5-17.

Table 5-17 Comparison of identical workload with different choice of locking controls

Run identification	A	B	C	D	E	F	
Sub-system	DB9A	DB9A	DB9A	DB9A	D9C1	D9C2	D9CG <sup>a</sup>
Data sharing	NO				YES		
Lock size	ROW	PAGE	Table space	ANY (PAGE)	ANY (PAGE)	ANY (PAGE)	
CURRENTDATA	YES			NO			
DB2 CPU Class 1	84.51	82.36	81.66	82.24	44.57	45.89	90.46
DB2 CPU Class 2	79.38	77.39	76.74	77.21	41.99	43.25	85.24
Timeouts	0	0	0	0	0	0	0
Deadlocks	0	0	0	0	0	0	0
Escal.(SHARED)	0	0	0	0	0	0	0
Escal.(EXCLUS)	0	0	0	0	0	0	0

<b>MAX PG/ROW LOCKS HELD</b>	228	20	2	20	20	25	45
<b>LOCK request</b>	683943	131267	52129	72938	46018	44243	90261
<b>UNLOCK request</b>	613992	65218	7338	8019	5349	4987	10336
<b>QUERY request</b>	0	0	0	0	0	2	2
<b>CHANGE request</b>	6699	6699	5871	6309	3288	3266	6554
<b>OTHER request</b>	0	0	0	0	0	0	0
<b>Total suspensions</b>	10	11	2	7	40	34	74
<b>Lock suspensions</b>	0	0	0	0	0	0	0
<b>IRLM LATCH suspensions</b>	10	11	2	7	8	10	18
<b>OTHER suspensions</b>	0	0	0	0	32	24	56

a. The D9CG column represents the sum of the values for the individual runs on the two members (D9C1 and D9C2) of the data sharing group.

The workload that was run to create the data shown in Table 5-17 on page 148 was created by running the Workload Generator. See “Stored procedures workload” on page 442 for a description. The workload was run single threaded for the non-data sharing environments, which is why there are no locking issues in terms of timeouts or deadlocks.

The profile of the work for the runs is shown in Table 5-18.

*Table 5-18 Workload SQL statement profile*

<b>Statement</b>	<b>Executions</b>
SELECT	60796
INSERT	17160
UPDATE	4286
DELETE	101
DESCRIBE	40
PREPARE	200
OPEN	900
FETCH	606945
CLOSE	740
<b>Total</b>	<b>691168</b>

Observations:

- ▶ Perhaps the most significant point is that the CPU cost differs only marginally for the different choices of locksize.
- ▶ The lock avoidance run (run D) takes only a few more locks than locksize table space (run C) and significantly less than locksize page with currentdata yes (run B).
- ▶ The cost of data sharing should be considered.

## 5.19 How to prevent locking problems

Locking problems originating in the application can be prevented if the application developers are aware of the locking concepts and the most common DB2 locking problems.

Application developers have to think about potential locking problems when participating in application and database design, developing programs, and also while constructing SQL statements. The application programmer must always try to minimize lock duration and lock suspension time without losing the logical integrity of data in the database. In most cases, the application programmer also has to allow for maximizing the concurrent access to data by users or programs. How much effort the application programmer has to put into these different issues also depends on the business needs of the environment.

Locks can cause situations like suspension, deadlock, timeout, and lock escalation that could have a substantial negative effect on application performance and functionality. The applications should be designed to promote concurrency and avoid locking issues with the understanding that other applications may need access to the same data at the same time.

### 5.19.1 Suspension, deadlock, timeout, and lock escalation

The adverse effects of locks that you want to avoid and minimize include suspension, deadlock, timeout, and lock escalation.

#### **Suspension**

An application process is suspended when it requests a lock that is already held by another application process and cannot be shared. The suspended process temporarily stops running. The suspended process resumes running when all the processes that hold the conflicting lock release it.

If the lock is not released (within a preset time), then the requesting process times out or deadlocks, and the process can only be resumed manually to deal with the deadlock or timeout condition.

#### **Timeout**

An application process is said to time out when it is terminated because it has been suspended for longer than a preset interval.

When a process gets a timeout, DB2 terminates the process, rolls back the updates, issues two messages to the console, and returns SQLCODE -911 or -913 to the process (SQLSTATES '40001' or '57033'). Reason code 00C9008E is returned in the SQLERRD(3) field of the SQLCA. Alternatively, you can use the GET DIAGNOSTICS statement to check the reason code.

#### **Effects of timeout in IMS**

If you are using IMS, and a timeout occurs, the following actions take place:

- ▶ In a DL/I batch application, the application process abnormally terminates with a completion code of 04E and a reason code of 00D44033 or 00D44050.
- ▶ In any IMS environment except DL/I batch:
  - DB2 performs a rollback operation on behalf of your application process to undo all DB2 updates that occurred during the current unit of work.



- For a non-message driven BMP, IMS issues a rollback operation on behalf of your application. If this operation is successful, IMS returns control to your application, and the application receives SQLCODE -911. If the operation is unsuccessful, IMS issues user abend code 0777, and the application does not receive an SQLCODE.
- For an MPP, IFP, or message driven BMP, IMS issues user abend code 0777, rolls back all uncommitted changes, and reschedules the transaction. The application does not receive an SQLCODE. COMMIT and ROLLBACK operations do not time out. The command STOP DATABASE, however, might time out and send messages to the console, but it retries up to 15 times.

## Deadlock

A deadlock occurs when two or more application processes each hold locks on resources that the others need and without which they cannot proceed. In some cases, a deadlock can occur if two application processes attempt to update data in the same page or table space.

DB2 initiates a DEADLOCK detection cycle at preset intervals (DEADLOCK TIME). When a DEADLOCK is identified, action is taken immediately, as there is no possibility for a DEADLOCK to be resolved without intervention. DB2 can either roll back the current unit of work for one of the processes or request a process to terminate. Either way, that frees the locks and allows the remaining processes to continue. Reason code 00C90088 is returned in the SQLERRD(3) field of the SQLCA. Alternatively, you can use the GET DIAGNOSTICS statement to check the reason code. (The codes that describe the exact DB2 response depend on the operating environment.) It is possible for two processes to be running on distributed DB2 subsystems, each trying to access a resource at the other location. In that case, neither subsystem can detect that the two processes are in deadlock; the situation resolves only when one process times out.

## Deadlocks and TSO, batch, and CAF

When a deadlock or timeout occurs in these environments, DB2 attempts to roll back the SQL for one of the application processes. If the ROLLBACK is successful, that application receives SQLCODE -911. If the ROLLBACK fails, and the application does not abend, the application receives SQLCODE -913.

## Deadlocks and IMS

If you are using IMS, and a deadlock occurs, the following actions take place:

- ▶ In a DL/I batch application, the application process abnormally terminates with a completion code of 04E and a reason code of 00D44033 or 00D44050.
- ▶ In any IMS environment except DL/I batch:
  - DB2 performs a rollback operation on behalf of your application process to undo all DB2 updates that occurred during the current unit of work.
  - For a non-message driven BMP, IMS issues a rollback operation on behalf of your application. If this operation is successful, IMS returns control to your application, and the application receives SQLCODE -911. If the operation is unsuccessful, IMS issues user abend code 0777, and the application does not receive an SQLCODE.
  - For an MPP, IFP, or message driven BMP, IMS issues user abend code 0777, rolls back all uncommitted changes, and reschedules the transaction. The application does not receive an SQLCODE.

## Deadlocks and CICS

If you are using CICS and a deadlock occurs, the CICS attachment facility decides whether or not to roll back one of the application processes, based on the value of the ROLBE or ROLBI parameter. If your application process is chosen for rollback, it receives one of two SQLCODEs in the SQLCA:

- ▶ -911 A SYNCPOINT command with the ROLLBACK option was issued on behalf of your application process. All updates (CICS commands and DL/I calls, as well as SQL statements) that occurred during the current unit of work have been undone. (SQLSTATE '40001')
- ▶ -913 A SYNCPOINT command with the ROLLBACK option was not issued. DB2 rolls back only the incomplete SQL statement that encountered the deadlock or timed out. CICS does not roll back any resources. Your application process should either issue a SYNCPOINT command with the ROLLBACK option itself or terminate. (SQLSTATE '57033') Consider using the DSNTIAC subroutine to check the SQLCODE and display the SQLCA. Your application must take appropriate actions before resuming.

### 5.19.2 Deadlock avoidance

Deadlocks tend to occur in hot spots. First, know where hot spots occur and consider correcting them to avoid deadlocks. See 4.6, “Hot spot scenarios” on page 80 for a discussion of hot pages and rows.

As a general rule, consider the following procedures to avoid deadlocks:

- ▶ Reduce the lock duration using the recommendations in 5.2, “Optimizing concurrency and locking” on page 100.
- ▶ Commit as frequently as possible using the recommendations in “Unit of recovery and unit of work” on page 120.
- ▶ Use ordered updates on different kinds of resources that cannot detect each other's deadlock situations, for example, between DB2 and IMS.
- ▶ Use ordered updates within the same kind of resources, for example, in a DB2 subsystem, although care should be taken in multi-table SQL statements in concurrent programs where the join order can change between binds.
- ▶ Consider row-level locking if different applications are accessing different rows.
- ▶ Code cursors unambiguously, specifically stating FOR FETCH ONLY when appropriate, therefore acquiring less restrictive locks, as discussed in “Lock avoidance” on page 33.

### Code FOR FETCH ONLY for read-only cursors

If you know that this cursor is used only for reading, notify DB2 so that it can take advantage of the situation. Do not code your cursor to be ambiguous if it does not need to be.

## Code FOR UPDATE OF for updateable cursors

Specifying FOR UPDATE OF in a cursor communicates to DB2 your intent to update or delete the rows that you read from the cursor. Figure 5-9 shows how coding the FOR UPDATE OF clause can avoid a deadlock situation.

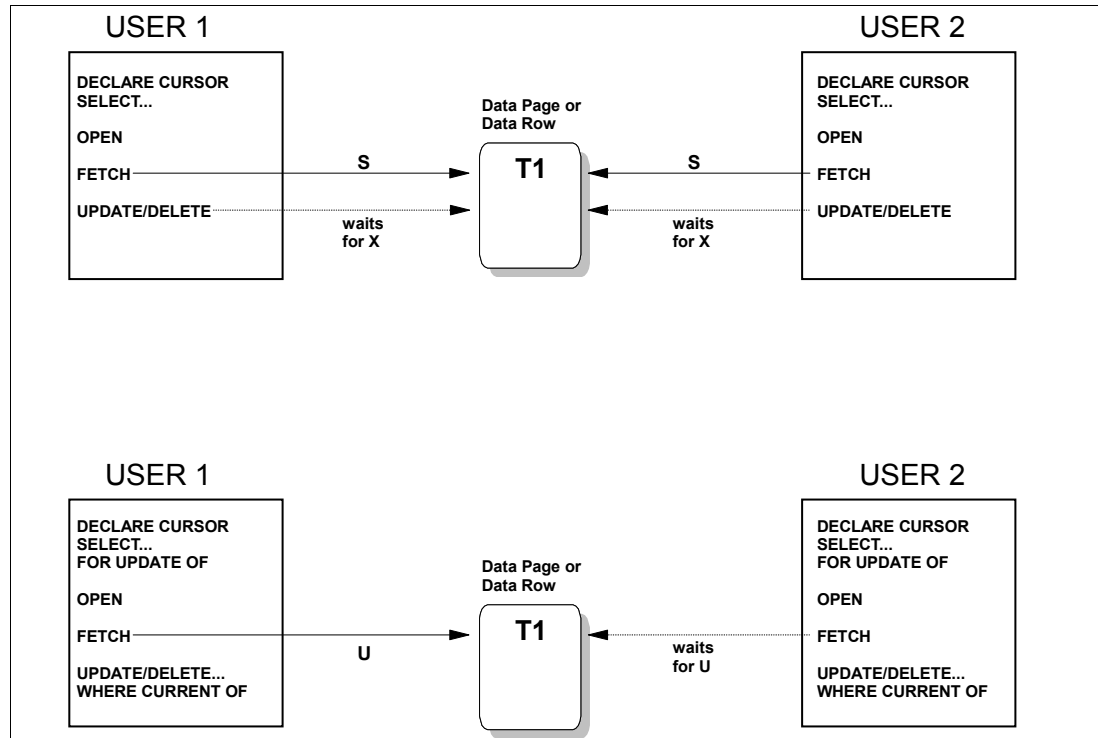


Figure 5-9 Using FOR UPDATE OF to acquire a U lock

The cursor in the top half of Figure 5-9 is coded without the FOR UPDATE OF clause. Both USER 1 and USER 2 fetch a row (which acquires an S lock) and then update or delete the row.

This creates a deadlock because:

- USER 1:
  - Has an S lock on T1 (data page or row)
  - Waits for an X lock on T1 (data page or row)
- USER 2:
  - Has an S lock on T1 (data page or row)
  - Waits for a X lock on T1 (data page or row)

The cursor in the bottom half of Figure 5-9 has the FOR UPDATE OF clause coded. Both USER 1 and USER 2 fetch a row and then update or delete the row.

There is no deadlock situation because U lock is used:

- User 1 has a U lock on T1 (data page or row).
- User 2 waits for a U lock on T1 (data page or row).

**Consideration:** This technique cannot be used when ordering is needed because the FOR UPDATE OF and the ORDER BY clauses cannot be specified in the same cursor.

## Deadlocks due to unordered updates

When different application processes perform update or delete operations in different sequences at the same time, it is likely that deadlocks occurs. Figure 5-10 shows two deadlocks caused by applications that:

- Update tables in a different order
- Update rows inside one table in a different order

These deadlocks can occur with both page level locking and row level locking.

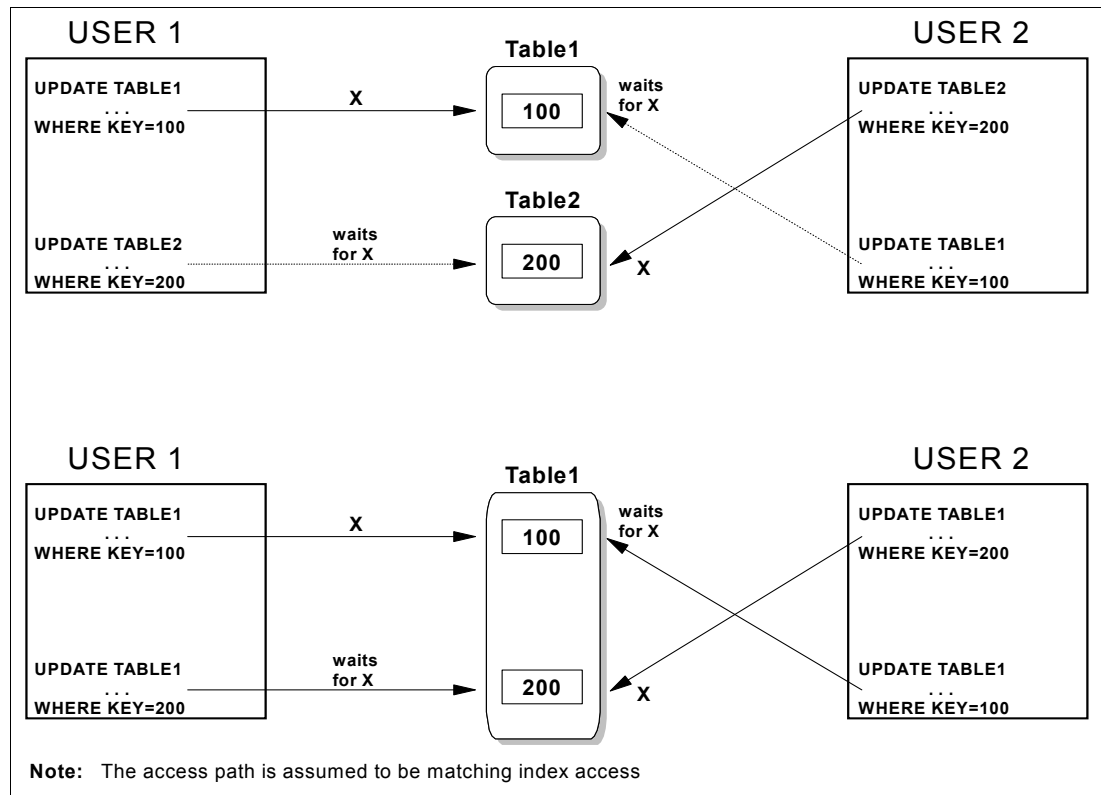


Figure 5-10 Deadlocks due to unordered updates

Deadlocks due to 'unordered updates and deletes can be avoided by:

- Ordered access to the different tables

For parent and dependent tables that will both be updated or deleted in an application process, the dependent table should be processed first. This is because DB2 enforced RI will require that the child rows are deleted first in the delete process.

In other cases, always using an arbitrary order, such as alphabetical order, might be appropriate.

- Ordered access to the rows within a table

Always performing updates and deletes in the order of the clustering index is one way to reduce the potential for deadlock and possibly achieve better performance from dynamic prefetch.

### 5.19.3 Application action after a deadlock or timeout

The typical SQLCODE returned to the victim of a deadlock or timeout is a -911, which indicates that the current unit of work has been rolled back to the last commit point and all locks have been released. In some environments, it is possible for the victim of a deadlock or timeout to receive an SQLCODE of -913, which indicates that no rollback has been performed.

### 5.19.4 Recommendations to prevent locking problems

The following application design recommendations can be used to prevent locking problems and promote concurrency:

- ▶ If you are using control tables to generate keys for tables, you will likely experience contention on those control tables, especially as your volume of work increases. Consider using IDENTITY columns or SEQUENCES to generate keys instead of control tables. Use sequence objects to generate unique, sequential numbers. Using an identity column is one way to generate unique sequential numbers. However, as a column of a table, an identity column is associated with and tied to the table, and a table can have only one identity column.

Your applications might need to use one sequence of unique numbers for many tables or several sequences for each table. As a user-defined object, sequences provide a way for applications to have DB2 generate unique numeric key values and to coordinate the keys across multiple rows and tables. The use of sequences can avoid the lock contention problems that can result when applications implement their own sequences, such as in a one-row table that contains a sequence number that each transaction must increment. If such a method is used, you should use `select from update counter=counter+ 1` and then issue a COMMIT as soon as possible.

With DB2 sequences, many users can access and increment the sequence concurrently without waiting. DB2 does not wait for a transaction that has incremented a sequence to commit before allowing another transaction to increment the sequence again.

- ▶ Applications should include retry logic. This standard technique is needed when other techniques are not perfect, as sometimes a retry after a deadlock or timeout would probably be successful and eliminate the need to initiate the process again.

Particularly, include logic in batch programs so that it retries an operation after a deadlock or timeout. Such a method could help you recover from the situation without assistance from operations personnel.

- Field SQLERRD(3) in the SQLCA returns a reason code that indicates whether a deadlock or timeout occurred.
  - Alternatively, you can use the GET DIAGNOSTICS statement to check the reason code.
- ▶ When multiple applications access the same data, try to make them do so in the same sequence so that you prevent situations that may cause deadlocks. For example, if two applications access five rows of data, make sure that both applications access rows 1, 2, 3, 4, and 5, in that order. In this case, the first application to access the data delays the second, but the two applications cannot deadlock. For the same reason, try to make different applications access the same tables in the same order.
- ▶ Consider using a technique called lock avoidance whenever possible. When you bind using option CURRENTDATA(NO), DB2 can test to see if a row or page has committed data on it. If it does, DB2 does not have to obtain a lock on the data at all.

- ▶ To avoid an unnecessary lock contention, issue a COMMIT statement as soon as possible after reaching a point of consistency, even in read-only applications. Statements issued through SPUFI can be committed immediately by the SPUFI autocommit feature. Taking commit points frequently in a long running unit of recovery has the following benefits at the possible cost of more CPU usage and log write I/Os:
  - Reduces lock contention, especially in a data sharing environment
  - Improves the effectiveness of lock avoidance, especially in a data sharing environment
  - Reduces the elapsed time for DB2 system restart following a system failure
  - Reduces the elapsed time for a unit of recovery to roll back following an application failure or an explicit rollback request by the application
  - Provides more opportunity for utilities, such as online REORG, to break in
- ▶ If you define a cursor using the WITH HOLD option, the locks it needs can be held past a commit point. Use the CLOSE CURSOR statement as soon as possible in your application to cause those locks to be released and the resources that they hold to be freed at the first commit point that follows the CLOSE CURSOR statement.
- ▶ Bind plans with the ACQUIRE(USE) option in most cases. ACQUIRE(USE), which indicates that DB2 acquires table and table space locks when the objects are first used and not when the plan is allocated, is the best choice for concurrency. Packages are always bound with ACQUIRE(USE) by default. ACQUIRE(ALLOCATE), which is an option for plans, but not for packages, can provide better protection against timeouts. Consider ACQUIRE(ALLOCATE) for applications that need gross locks instead of intent locks or that run with other applications that might request gross locks instead of intent locks. Acquiring the locks at plan allocation also prevents any one transaction in the application from incurring the cost of acquiring the table and table space locks. If you need ACQUIRE(ALLOCATE), you might want to bind all DBRMs directly to the plan.
- ▶ Bind applications with the ISOLATION(CS) and CURRENTDATA(NO) options in most cases. If you do not use ISOLATION(CS) and CURRENTDATA(NO), in order of decreasing preference for concurrency, use the bind options:
  - ISOLATION(CS) with CURRENTDATA(YES), when data returned to the application must not be changed before your next FETCH operation.
  - ISOLATION(RS), when data returned to the application must not be changed before your application commits or rolls back. However, you do not care if other application processes insert additional rows.
  - ISOLATION(RR), when data evaluated as the result of a query must not be changed before your application commits or rolls back. New rows cannot be inserted into the answer set.
- ▶ Use ISOLATION(UR) option cautiously. The Resource Recovery Services attachment facility UR isolation acquires almost no locks on rows or pages. It is fast and causes little contention, but it reads uncommitted data. Do not use it unless you are sure that your applications and users can accept the logical inconsistencies that can occur. As an alternative, consider using the SKIP LOCKED data option if omitting data is preferable to reading uncommitted data in your application.
- ▶ Consider coding the USE and KEEP UPDATE LOCKS option in your SELECT statements, when you want to prevent data integrity exposure. Using this option on cursor select directly acquires an X lock; no lock promotion is required when data is to be updated subsequently. This option also reduces the possibility of deadlocks.

- If you have many static code tables that are used across different applications, we recommend that you consider alternatives to allow each application its own copy of the data. One alternative is to use a declared temporary table (DTT) to store the information from the code table. This technique is especially useful for long running applications that need to load the code table information once, but it will use it throughout the length of the job. Using a DTT reduces contention because the only locking that occurs on the code table will be at the time it is read to populate the DTT. Once the DTT is populated, you can use SQL to access the codes without requiring locks against the base code table.

Note that if the program using the DTT needs to be rerun later, the content populated into the DTT could be different on the second run. If that is not acceptable, then a real table with persistent data needs to be created.

- Examine multi-row operations, such as multi-row inserts, positioned updates, and positioned deletes, which have the potential of expanding the unit of work. This situation can affect the concurrency of other users that access the data. You can minimize contention by adjusting the size of the host-variable-array, committing between inserts, updates, and preventing lock escalation.
- Use global transactions. The Resource Recovery Services attachment facility (RRSAF) relies on a z/OS component called Resource Recovery Services (RRS). RRS provides system-wide services for coordinating two-phase commit operations across z/OS products. For RRSAF applications and IMS transactions that run under RRS, you can group together a number of DB2 agents into a single global transaction. A global transaction allows multiple DB2 agents to participate in a single global transaction and thus share the same locks and access the same data. When two agents that are in a global transaction access the same DB2 object within a unit of work, those agents do not deadlock or timeout with each other. The following restrictions apply:
  - Parallel Sysplex® is not supported for global transactions.
  - Because each of the branches of a global transaction are sharing locks, uncommitted updates issued by one branch of the transaction are visible to other branches of the transaction.
  - Claim and drain processing is not supported across the branches of a global transaction, which means that attempts to issue CREATE, DROP, ALTER, GRANT, or REVOKE might deadlock or time out if they are requested from different branches of the same global transaction.
  - LOCK TABLE might deadlock or time out across the branches of a global transaction.
- Use optimistic concurrency control. Optimistic concurrency control represents a faster, more scalable locking alternative to database locking for concurrent data access. It minimizes the time for which a given resource is unavailable for use by other transactions. When an application uses optimistic concurrency control, locks are obtained immediately before a read operation and released immediately. Update locks are obtained immediately before an update operation and held until the end of the transaction. Optimistic concurrency control uses the row change token to test whether data has been changed by another transaction since the last read operation. Because DB2 can determine when a row was changed, you can ensure data integrity while limiting the time that locks are held. With optimistic concurrency control, DB2 releases the row or page locks immediately after a read operation. DB2 also releases the row lock after each FETCH, taking a new lock on a row only for a positioned update or delete to ensure data integrity.

After you establish a row change time stamp column, DB2 maintains the contents of this column. When you want to use this change token as a condition when making an update, you can specify an appropriate predicate for this column in a WHERE clause.

- Avoid optimistic concurrency control if there is a possibility of concurrent updates that could conflict with your updates. In such cases, the read followed by searched update approach should be converted to SELECT ... FOR UPDATE of..., followed by a positioned update. This non-optimistic concurrency control would also eliminate any data integrity exposure issues.
- Applications should intercept error information such as deadlock and timeout for better problem diagnosis. Also, the development community should get subsystem level information pertaining to locking periodically and use it to redesign the application.

Other recommendations in the areas of database design and system parameters are discussed in Chapter 4, “Database design considerations” on page 65 and Chapter 7, “System considerations” on page 201, respectively.

The flow chart in Figure 5-11 describes the lock escalation scenario and how DB2 uses NUMLKTS and LOCKMAX values. by the same token, you can develop an application specific mechanism to control lock escalation by using the NUMLKTS parameter at the system level and LOCKMAX parameter at the table space level.

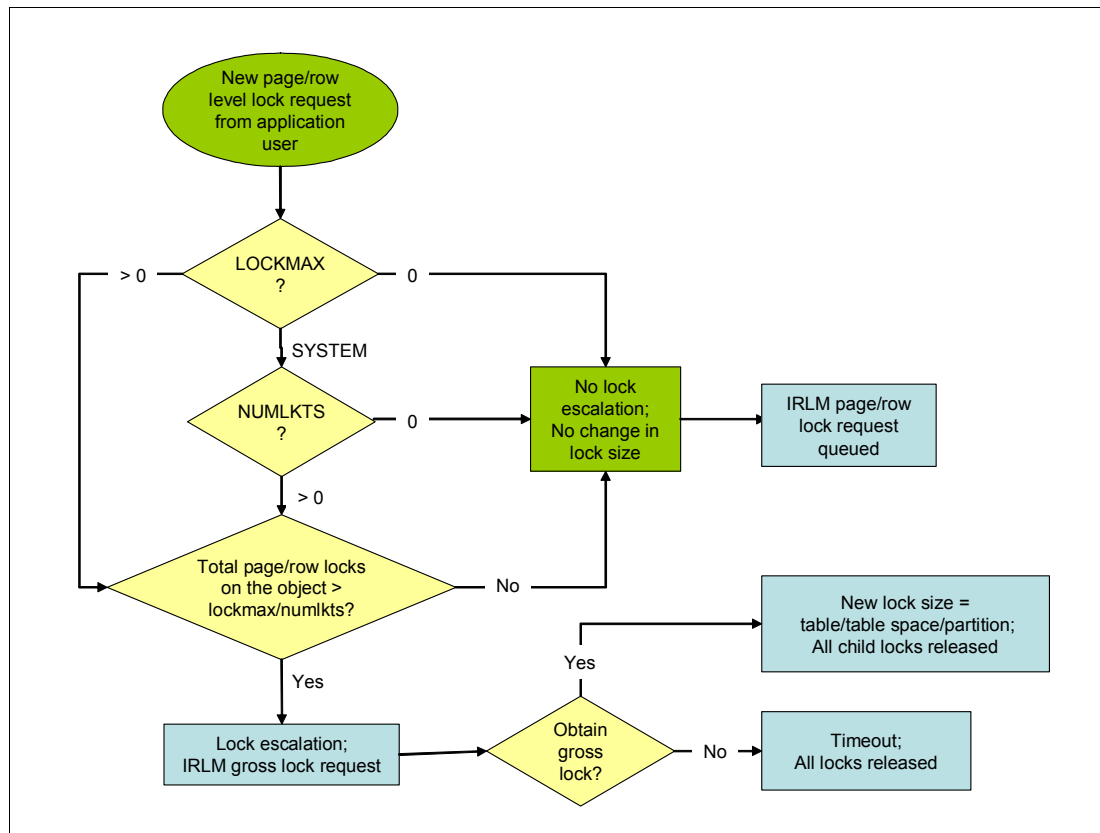


Figure 5-11 Lock escalation controls

Deadlock and timeout problem determination is discussed in 8.1, “Different types of locking and concurrency problems” on page 220.

Always remember that the IRLM lock requests are serialized on a first come, first served basis. Refer to 3.3.1, “IRLM principles of operation” on page 45 before designing your application for concurrency.



Other important points that you should consider when you work with DB2 locks include:

- ▶ You usually do not have to lock data explicitly in your program.
- ▶ DB2 ensures that your program does not retrieve uncommitted data unless you specifically code for that.
- ▶ Any page or row where your program updates, inserts, or deletes stays locked at least until the end of a unit of work, regardless of the isolation level. No other process can access the page or row in any way until then, unless you specifically allow that access to that process.
- ▶ Commit often for concurrency.
- ▶ Bind with ACQUIRE(USE) to improve concurrency.
- ▶ Set ISOLATION (usually RR, RS, or CS) when you bind the plan or package.
  - With RR (repeatable read), all accessed pages or rows are locked until the next commit point.
  - With RS (read stability), all qualifying pages or rows are locked until the next commit point.
  - With CS (cursor stability), only the pages or rows currently accessed can be locked, and those locks might be avoided. (You can access one page or row for each open cursor.)
- ▶ You can also set isolation for specific SQL statements, using WITH.
- ▶ A deadlock can occur if two processes each hold a resource that the other needs.
- ▶ In a deadlock, one process is chosen as “victim”; its unit of work is rolled back, and an SQL error code is issued.
- ▶ You can lock an entire non segmented table space, or an entire table in a segmented table space, with the LOCK TABLE statement.

## 5.20 List of common lock types: OMEGAMON PE online monitor

Table 5-19 lists the common lock types used by DB2, as abbreviated and used by the OMEGAMON PE Version 4.2 online monitoring tool. This table can be used to interpret some of the examples shown in this chapter that contain a OMEGAMON PE online monitor snapshot list of locks acquired by DB2.

*Table 5-19 List of common lock types used in OMEGAMON PE V4.2*

Lock type	Short description	Long description
ALBP	Alter buffer pool	A lock on a buffer pool during lock execution of an ALTER BUFFERPOOL command.
BIND	Bind lock	An autobind/remote bind lock.
CCAT	CATMAINT convert	This lock is acquired when catalog lock catalog conversion is performed.
CDIR	CATMAINT convert	This lock is acquired when a directory lock directory conversion is performed.
CDRN	Cursor Stability	This lock is acquired to drain all drain locks for CS read access to an object.

Lock type	Short description	Long description
CMIG	CATMAINT	This lock is acquired when a migration lock catalog migration is performed.
COLL	Collection lock	A collection lock.
DBEX	Database exception LPL/GRECP lock	A lock on either a “Logical LPL/GRECP lock page list” or “Group buffer pool recovery pending” database exception status. Used only in a data sharing environment.
DBXU	Database exception	A lock used in a table space update lock for updating database exception.
DPAG	Data page lock	A page lock in a table space. When programs read or update data, they acquire a page lock containing the data.
DSET	Partitioned lock	A partitioned table space is created when you create a table space using the SQL statement CREATE TABLESPACE with the Numparts parameter. Only one table can be stored on a partitioned table space. Each partition contains one part of a table. The partitioned lock only locks the partition with the data being referenced.
DTBS	Database lock	A lock on the database.
GRBP	Group BP start/stop lock	A group buffer pool start/stop lock. Used only in a data sharing environment.
HASH	Hash anchor lock	A hash anchor lock.
IEOF	Index EOF	A lock acquired at index end of file.
MDEL	Mass delete lock	This lock is acquired when doing a mass delete from a table (for example, when you DELETE FROM a table) within a segmented table space. It is used to prevent another user from reusing freed segments before a delete operation is committed.
PALK	Partition Level lock	A lock exists at the partition locking level of the database.
PBPC	Group BP level castout P-lock	A physical lock acquired when a castout P-lock of a group buffer pool occurs. Castout is the process of writing pages in the group buffer pool out to DASD. Used only in a data sharing environment.
PCDB	DDF CDB P-lock	A Distributed Data Facility communication database physical lock. Used only in a data sharing environment.
PDBD	PDBD P-lock	A database descriptor physical lock. Used only in a data sharing environment.
PDSO	Pageset or partitioned pageset open lock	If the data set supporting the partitioned table space that is referenced pageset open by the application is not lock opened, the program will acquire a lock to open the data set. The data set will stay open if CLOSE=NO is defined in the SQL statement creating the table space.
PITR	Index manager tree P-lock	An index manager tree physical tree P-lock lock. Used only in a data sharing environment.
PPAG	Page P-lock	A physical lock on a page. Used only in a data sharing environment.



## Utilities, commands, and SQL

Claims and drains provide a serialization mechanism to control the concurrency of some DB2 resources between SQL processes, utilities, and commands. In addition, restricted states are set by DB2 utilities or DB2 commands, which could severely affect concurrency and availability of the affected DB2 objects.

This chapter discusses how DB2 handles serialization between

- ▶ DB2 utilities and SQL statements
- ▶ Interaction between some common DB2 commands and SQL statements
- ▶ How DML and DDL can conflict with each other

The following topics are covered:

- ▶ Claims and drains for concurrency control
- ▶ Serialization between utilities and SQL
- ▶ Deadlock avoidance between utilities and SQL
- ▶ Partition independence
- ▶ Online REORG best practices for concurrency
- ▶ Online utilities and concurrency
- ▶ DB2 command considerations
- ▶ Locks on DB2 objects
- ▶ Restricted states

## 6.1 Claims and drains for concurrency control

Claims and drains are taken at the object level only and not at the page or row level. DB2 utilities, commands, and some ALTER, CREATE, and DROP statements can take over access to some DB2 objects independent of any transaction locks that are held on the object.

### Claims and drains on an index

Unlike transaction locks, the claims and drains can be taken on indexes also. DB2 utilities, commands, and some ALTER, CREATE, and DROP statements can take over access to certain objects.

The following objects are subject to takeover by those operations:

- ▶ Simple and segmented table spaces
- ▶ Partitions of table spaces
- ▶ LOB table spaces
- ▶ XML table spaces
- ▶ Non-partitioned index spaces
- ▶ Partitions of index spaces
- ▶ Logical partitions of non-partitioned indexes

### 6.1.1 Claim

A claim is a notification to DB2 that an object is being accessed.

When an application first accesses an object, within a unit of work, it makes a claim on the object. It releases the claim at the next commit point. Unlike a transaction lock, a claim normally does not persist past the commit point. To access the object in the next unit of work, the application must make a new claim.

However, an exception exists: If a cursor defined with the WITH HOLD clause is positioned on the claimed object, the claim is not released at a commit point.

A claim indicates activity on or interest in a particular page set or partition to DB2. Claims prevent drains from occurring until the claim is released.

### Three classes of claims

Table 6-1 shows the three classes of claims and the actions that they allow.

*Table 6-1 Claim classes and how DB2 uses them*

Claim class	Notification to DB2 that the following action is being performed on the object involved
Write	Reading, updating, inserting, and deleting
Repeatable read	Reading only, with repeatable read (RR) isolation
Cursor stability read	Reading only, with read stability (RS), cursor stability (CS), or uncommitted read (UR) isolation

## 6.1.2 Drains

A drain is the action of taking over access to an object by preventing new claims and waiting for existing claims to be released.

A utility can drain a partition when applications are accessing it. The drain quiesces the applications by allowing each one to reach a commit point, but preventing any of them, or any other applications, from making a new claim. When no more claims exist, the process that drains (the drainer) controls access to the drained object. The applications that were drained can still hold transaction locks on the drained object, but they cannot make new claims until the drainer has finished.

### Drained claim classes

A drainer does not always need complete control. It could drain the following combinations of claim classes:

- ▶ Only the write claim class
- ▶ Only the repeatable read claim class
- ▶ All claim classes

For example, the CHECK INDEX utility needs to drain only writers from an index space and its associated table space. The RECOVER TABLESPACE utility, however, must drain all claim classes from its table space. The REORG utility can drain either writers (with DRAIN WRITERS) or all claim classes (with DRAIN ALL).

### How DB2 uses drain locks

A drain lock prevents conflicting processes from trying to drain the same object at the same time.

Processes that drain only writers can run concurrently, but a process that drains all claim classes cannot drain an object concurrently with any other process. To drain an object, a drainer first acquires one or more drain locks on the object, one for each claim class that it needs to drain. When the locks are in place, the drainer can begin after all processes with claims on the object have released their claims. A drain lock also prevents new claimers from accessing an object while a drainer has control of it.

### Types of drain locks

Three types of drain locks on an object correspond to the three claim classes:

- ▶ Write
- ▶ Repeatable read
- ▶ Cursor stability read

In general, after an initial claim has been made on an object by a user, no other user in the system needs a drain lock. When the drain lock is granted, no drains on the object are in process for the claim class needed, and the claimer can proceed.

The claimer of an object requests a drain lock in the following exceptional case: The claim is the first claim on an object before its data set has been physically opened. Here, acquiring the drain lock ensures that no exception states prohibit allocating the data set.

For data sharing, when the claimer gets the drain lock, it makes its claim and releases the lock before beginning its processing.

For non-data-sharing, exception states are checked when the pageset control blocks are built (no drain locks needed) and the “drain in control” status is set accordingly. So, the first claim request may not need to get the drain lock.

A drain on the object is in process for the claim class needed. In this case, the claimer waits for the drain lock.

## 6.2 Utility concurrency and compatibility

Refer to the “Concurrency and compatibility” topic for each utility in the *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.

### 6.2.1 Utility locks on the catalog and directory

When the target of a utility is an object in the catalog or directory, such as a catalog table, the utility either drains or claims the object. When the target is a user-defined object, the utility claims or drains it, but also uses the directory and, perhaps, the catalog, for example, to check authorization. In those cases, the utility uses transaction locks on catalog and directory tables. It acquires those locks in the same way as an SQL transaction does. For information about the SQL statements that require locks on the catalog, refer to 6.7.2, “Contention on the DB2 catalog” on page 192.

#### The UT SERIAL lock

Access to the SYSUTILX table space in the directory is controlled by a unique lock called UT SERIAL. A utility must acquire the UT SERIAL lock to read or write in SYSUTILX, whether SYSUTILX is the target of the utility or is used only incidentally.

### 6.2.2 Compatibility of utilities

Two utilities are considered compatible if they do not need access to the same object at the same time in incompatible modes.

The concurrent operation of two utilities is not typically controlled by either drain locks or transaction locks, but merely by a set of compatibility rules. Before a utility starts, it is checked against all other utilities running on the same target object. The utility starts only if all the others are compatible.

The check for compatibility obeys the following rules:

- ▶ The check is made for each target object, but only for target objects. Typical utilities access one or more table spaces or indexes, but if two utility jobs use none of the same target objects, the jobs are always compatible. An exception is a case in which one utility must update a catalog or directory table space that is not the direct target of the utility. For example, the LOAD utility on a user table space updates DSNDB06.SYSCOPY. Therefore, other utilities that have DSNDB06.SYSCOPY as a target might not be compatible.
- ▶ Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions in the same table or index space are compatible.

- When two utilities access the same target object, their most restrictive access modes determine whether they are compatible. For example, if utility job 1 reads a table space during one phase and writes during the next, it is considered a writer. It cannot start concurrently with utility 2, which allows only readers on the table space. (Without this restriction, utility 1 might start and run concurrently with utility 2 for one phase, but then it would fail in the second phase, because it could not become a writer concurrently with utility 2.)

For details about which utilities are compatible, refer to each utility's description in *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.

## 6.3 Serialization between utilities and SQL

While SQL applications use transaction locks to control concurrent access to DB2 objects, DB2 utilities can obtain access to DB2 objects through claims, drains, and their own set of compatibility rules.

Online utilities allow other utilities and SQL statements to run concurrently on the same target object. In general, utilities detect claimers and wait (except for some online utilities).

For example, an application using UR isolation cannot run concurrently with a utility that drains all claim classes. For example, if CHECK DATA utility is running with the DELETE YES option, then no concurrent SQL access is allowed, including SELECT ... WITH UR on the target table.

### 6.3.1 Scenario for concurrency of utilities and SQL

Drains and claims are used to serialize between utilities and SQL. Figure 6-1 illustrates a situation where SQL applications and utilities coexist.

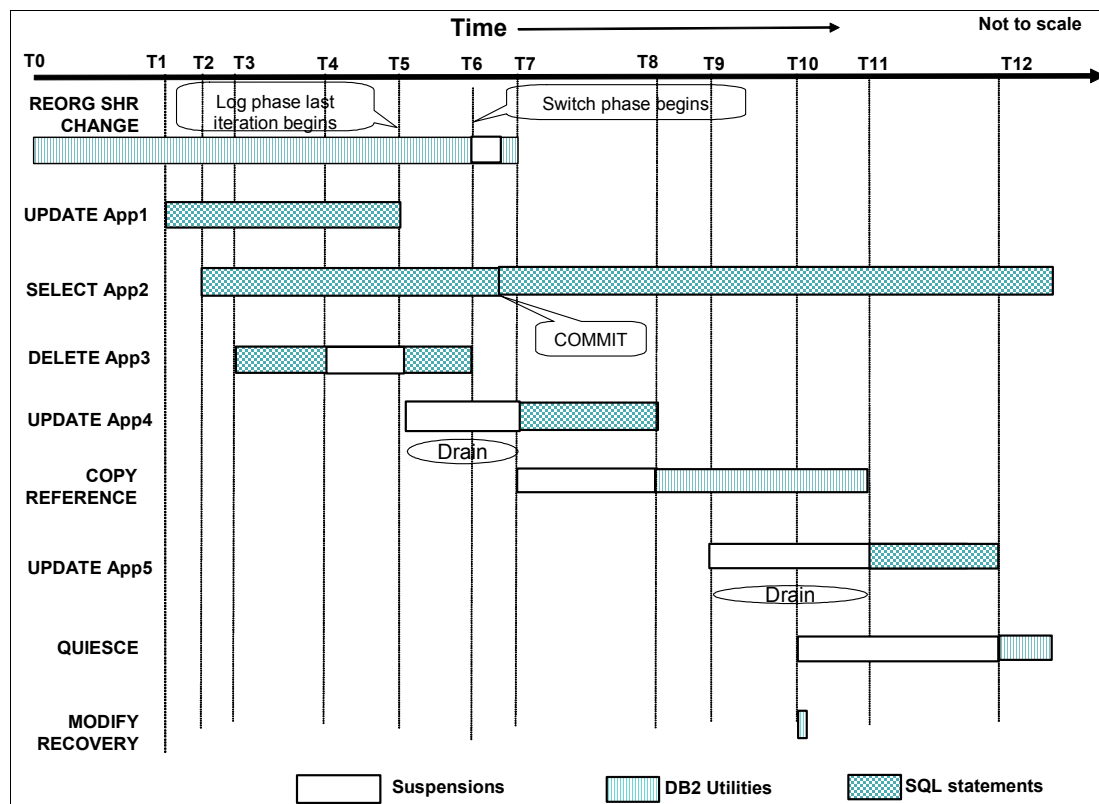


Figure 6-1 Concurrency of transactions and utilities

In our case, during a certain period of time, four SQL programs run concurrently and access the same table space partition with three different utilities. All claim, drain, and inter-utility compatibility mechanisms play a role in the example.

Here are the descriptions of the various time slices:

- Time T0: The online REORG TABLESPACE PART 1 SHRLEVEL CHANGE utility starts running and it claims the read claim class during unload phase. During the last iteration of the LOG phase, the utility will drain the write claim class, which allows concurrent access for SQL readers only. During the SWITCH phase, REORG drains all the claim classes during which no concurrent SQL access is allowed.

**Reminder:** The SHRLEVEL CHANGE REORG option, DRAIN WRITERS, which is the default, causes a drain of writers during the log phase when MAXRO is reached and a drain all during the switch phase.

The SHRLEVEL CHANGE option claims the read claim class on the partition during the UNLOAD phase, but does not claim any write claim class. So, at time T0, the write claim count on Partition 1 of the target table space is zero and the CS read claim count on Partition 1 of the target table space is 1. This read claim will be release just before the start of the SWITCH phase.



Table 6-2 shows the table space (Part 1) lock mode and claim counts at time=T0.

Table 6-2 Table space (Part 1) lock mode and claim counts at time=T0

Write claim	CS Read claim	Lock mode
0	1	None

You can also run more than one REORG SHRLEVEL CHANGE job concurrently on different partitions of the same table space, but only if the table space does not have any non-partitioned indexes (NPIs).

- Time T1: An UPDATE Application (App1) executes the following SQL statement by taking a write claim on Partition 1 of the target table space and locking the qualifying pages:

```
UPDATE RAVI.GLWTEMP
SET BONUS = 10
WHERE EMP_NO = 68516
WITH CS
```

In general, writers (an update statement in this case) make an implicit read claim, which can be either a CS or RR read claim (it will be a RR read claim if WITH RR is specified). Here, App1 makes an implicit CS read claim because the UPDATE statement has the WITH CS clause. The write claim count on Partition 1 of the target table space is 1 and the CS read claim count on Partition 1 of the target table space is 2. DB2 would acquire an IX lock at the partition level.

Table 6-3 shows the table space (Part 1) lock mode and claim counts at time=T1.

Table 6-3 Table space (Part 1) lock mode and claim counts at time=T1

Write claim	CS Read claim	Lock mode
1	2	IX (Held by App1)

At this time, there is no conflict in resource usage, so both the REORG and the UPDATE processes run concurrently without any problems.

- Time T2: A read-only application (SELECT App2) executes the following SQL statement (through a cursor defined without HOLD option) by taking a CS read claim on Partition 1 of the target table space and uses lock avoidance techniques to avoid IRLM locking. This is a long running batch application with intermediate commits. The statement is as follows:

```
SELECT FIRSTNME, LASTNAME, HIREDATE
FROM RAVI.GLWTEMP
WHERE WORKDEPT = 30
FOR FETCH ONLY
WITH CS;
```

The write claim count on Partition 1 of the target table space is 1 and the CS read claim count on Partition 1 of the target table space is 3.

Table 6-4 shows the table space (Part 1) lock mode and claim counts at time=T2.

Table 6-4 Table space (Part 1) lock mode and claim counts at time=T2

Write claim	CS Read claim	Lock mode	
		App2	App1
1	3	IS	IX

Even if the SQL was coded with isolation UR (uncommitted read), the claim count will still be incremented by 1.

Until the REORG utility is entering the LOG phase (Time T5 in this example), any number of updaters can run concurrently with REORG. Though two updaters may contend for the same page (or row if row level locking is in place) and may cause timeout or suspension, but that would not have any thing to do with the concurrent online REORG utility.

- Time T3: A DELETE Application (App 3) executes the following SQL statement by making a write claim on Partition 1 of the target table space and locking some of its pages:

```
DELETE
FROM RAVI.GLWTEMP
WHERE WORKDEPT = 70
WITH CS;
```

The DELETE statement makes an implicit CS read claim because the SQL has the WITH CS clause. At this time, the write claim count on Partition 1 of the target table space is 2 and the CS read claim count on Partition 1 of the target table space is 4.

Table 6-5 shows the table space (Part 1) lock mode and claim counts at time=T3.

*Table 6-5 Table space (Part 1) lock mode and claim counts at time=T3*

Write claim	CS Read claim	Lock mode		
		App3	App2	App1
2	4	IX	IS	IX

- Time T4: The DELETE Application (App3) is suspended because App1 is holding an X lock on a page that is needed in App3. This is a normal locking conflict and has nothing to do with the concurrently running utility.

At this time, there is no change in the write claim count or CS read claim count on Partition 1 of the target table space. Only the locks on pages have changed at this time compared to that of T3.

Table 6-6 shows the table space (Part 1) lock mode and claim counts at time=T4.

*Table 6-6 Table space (Part 1) lock mode and claim counts at time=T4*

Write claim	CS Read claim	Lock mode		
		App3	App2	App1
2	4	IX	IS	IX

- Time T5: App1 successfully completes (it commits all the changes), freeing all claims and page/row locks that it had held. At this time, the write claim count on Partition 1 of the target table space is 1, and the CS read claim count on Partition 1 of the target table space is 3. App13, which was suspended, resumes its execution.

Table 6-7 shows the table space (Part 1) lock mode and claim counts at time=T5.

*Table 6-7 Table space (Part 1) lock mode and claim counts at time=T5*

Write claim	CS Read claim	Lock mode	
		App3	App2
1	3	IX	IS

At the same time (T5), the REORG utility enters the last iteration of the LOG phase. This is followed by another UPDATE process (App4) trying to execute the following SQL statement:

```
UPDATE RAVI.GLWTEMP
SET WORKDEPT = 20
WHERE FIRSTNME = 'THARUN'
WITH CS;
```

But, since the REORG utility now has a drain lock on the write claim classes, App4 is suspended.

- Time T6: App3 successfully completes (it commits all the changes), freeing all locks and claims that it held (the only write claim on this partition is released). The online REORG utility completes the last iteration of the LOG phase and goes to SWITCH phase, which starts to drain all claim classes. Before draining all the claims (at the beginning of the SWITCH phase), the claim acquired at the beginning of the REORG is released by DB2.

Now, the write claim count on Partition 1 of the target table space is 0, and the CS read claim count on Partition 1 of the target table space is 1. At this time, the only read claim is from the App2, so the SWITCH phase of the online REORG will wait for App1 to release the read claim.

Table 6-8 shows the table space (Part 1) lock mode and claim counts at time=T6.

*Table 6-8 Table space (Part 1) lock mode and claim counts at time=T6*

Write claim	CS Read claim	Lock mode	
		App4	App2
0	1	IX (wait)	IS

Subsequently, between T6 and T7, App2 issues a COMMIT and the REORG will break in to take over the table space through its drain locks (now the REORG will have three drain locks on this partition, namely WRITE drain lock, CS-READ drain lock, and RR-READ drain lock).

App4 would not be able to resume at this time because the online REORG utility is still holding the WRITE drain lock (along with the CS and RR read drain locks).

- Time T7: Online REORG completes successfully, freeing all the drain locks that it acquired in the LOG phase and SWITCH phase.

Even though the online REORG takes an inline image copy, let us assume that there is a need for a clean SHRLEVEL REFERENCE copy in this environment.

At T7, the COPY SHRLEVEL REFERENCE utility job (on Partition 1 of the same table space) is submitted for execution. Although it allows concurrent SQL read claimers, it has to drain all write claimers before starting the actual copy process.

Immediately after the completion of online REORG utility, App4 resumes its execution, as it was in the IRLM queue ahead of the image copy utility job, by taking a write claim on Partition 1 of the target table space, and also locking the qualifying pages. The UPDATE also makes an implicit CS read claim because the UPDATE statement had the WITH CS clause (refer the UPDATE statement under events discussion for Time T5 above).

Since the write claim count on Partition 1 of the target table space is 1 because of App4, the image copy utility goes into a suspended state (and waits to obtain a WRITE drain lock).

Now, the write claim count on Partition 1 of the target table space is 1, and the CS read claim count on Partition 1 of the target table space is 1.

Table 6-9 shows the table space (Part 1) lock mode and claim counts at time=T7.

Table 6-9 Table space (Part 1) lock mode and claim counts at time=T7

Write claim	CS Read claim	Lock mode	
		App4	App2
1	1	IX	IS

App2 is still holding the IS lock on partition 1 of the table space, as it was bound with RELEASE(DEALLOCATE), even though there are not any page level locks or any claims held by App2 at this time.

- Time T8: App4 successfully completes (it commits all the changes), freeing all locks and claims that it held. The image copy utility gets its drain lock on the writers on Partition 1 and starts the copy process. The utility makes an IX drain lock. The write claim count on Partition 1 of the target table space is 0, and the CS read claim count on Partition 1 of the target table space is 0 at this time.

Table 6-10 shows the table space (Part 1) lock mode and claim counts at time=T8.

Table 6-10 Table space (Part 1) lock mode and claim counts at time=T8

Write claim	CS Read claim	Lock mode	
		COPY utility	App2
0	0	IX (drain lock)	IS

- Time T9: App5 (UPDATE application) requests an S drain lock on Partition 1 of the table space and gets suspended because the S drain lock is not compatible with the IX drain lock held by image copy utility. The SQL being executed is:

```
UPDATE USER.STAFF
SET COMM = 20
WHERE WORKDEPT = 111
WITH CS;
```

Table 6-11 shows the table space (Part 1) lock mode and claim counts at time=T9.

Table 6-11 Table space (Part 1) lock mode and claim counts at time=T9

Write claim	CS Read claim	Lock mode		
		COPY Utility	App2	App5
0	0	IX (drain lock)	IS	IX (waits for COPY utility to complete)

At this time, there is no change in the write claim count or CS read claim count on Partition 1 of the target table space.

- Time T10: A QUIESCE TABLESPACE PART1 utility starts. Because recovering objects to a quiesce point will be faster because no work has to be backed out, the QUIESCE TABLESPACE PART1 utility runs periodically in this environment.

A QUIESCE utility drains all the write claim classes on Partition 1 of the target table space. Although this request is compatible with the IX drain lock already held on Partition 1 of the target table space, IRLM queues this request behind the S drain lock request of App5. IRLM serves requests from two different users on a “first come, first out” basis.

At the same time, the MODIFY RECOVERY TABLESPACE utility is started on Partition 1. This utility is not compatible with the image COPY utility, and as a result it fails immediately. In this case, the only remedy is to run the MODIFY RECOVERY utility after the successful completion of the COPY utility. For details about which utilities are compatible with COPY TABLESPACE utility, refer to *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.

The write claim count on Partition 1 of the target table space is 0, and the CS read claim count on Partition 1 of the target table space is 0.

Table 6-12 shows the table space (Part 1) lock mode and claim counts at time=T10.

Table 6-12 Table space (Part 1) lock mode and claim counts at time=T10

Write claim	CS Read claim	Lock mode			
		COPY Utility	App2	App5	QUIESCE
0	0	IX (drain lock)	IS	IX (waits for COPY utility)	IX (waits for App5)

- Time T11: Image copy utility completes. App5, which was suspended, gets the S drain lock on Partition 1 of the target table space and resumes. The UPDATE statement makes a WRITE claim and also an implicit CS read claim because the UPDATE statement had the WITH CS clause.

Now, the write claim count on Partition 1 of the target table space is 1 and the CS read claim count on Partition 1 of the target table space is 1.

Table 6-13 shows the table space (Part 1) lock mode and claim counts at time=T11.

Table 6-13 Table space (Part 1) lock mode and claim counts at time=T11

Write claim	CS Read claim	Lock mode		
		App2	App5	QUIESCE
1	1	IS	IX	IX (waits for App5)

- Time T12: App5 commits and completes, freeing all claims and locks that it held, after which the write claim count on Partition 1 of the target table space is 0, and the CS read claim count on Partition 1 of the target table space is also 0. The QUIESCE utility, which was queued behind App5, gets the IX drain lock on Partition 1 of the table space and starts executing. The QUIESCE utility can run concurrently with any number of read-only applications.

Table 6-14 shows the table space (Part 1) lock mode and claim counts at time=T12.

Table 6-14 Table space (Part 1) lock mode and claim counts at time=T12

Write claim	CS Read claim	Lock mode	
		App2	QUIESCE
0	0	IS	IX (drain lock)

### Potential timeouts in this scenario

The potential timeouts in the scenario described in 6.3.1, “Scenario for concurrency of utilities and SQL” on page 166 are:

- ▶ Time T4 to T5: During this period of time, App3 is waiting for a page that is held by application1. Assuming an installation IRLM Resource Wait Timeout value of 30 and Deadlock Time value of 1, the maximum time App3 can be suspended without getting a timeout is 31 seconds.
- ▶ Time T7 to T8: Assuming an installation IRLM Resource Wait Timeout value of 30, Deadlock Time value of 1, and Utility Timeout value of 6, the maximum time the image copy utility waits without being timed out is  $30 \text{ seconds} * 6 + 2 = 182 \text{ seconds}$ .
- ▶ Time T9 to T11: During this time period, application 5 is waiting for an S drain lock, which is incompatible with the IX drain lock held by the image copy utility. Assuming an installation IRLM Resource Wait Timeout value of 30 and Deadlock Time value of 1, the maximum time App5 can be suspended without being backed out is 31 seconds.
- ▶ Time T10 to T12: During this time period, a QUIESCE utility is waiting behind application 5 for the IX drain lock on Partition 1 of the target table space. Assuming an installation IRLM Resource Wait Timeout value of 30, Deadlock Time value of 1, and Utility Timeout value of 6, the maximum time Utility 2 waits without being timed out is 182 seconds.
- ▶ Time T5 to T6 and Time T6 to T7: The TIMEOUT parameter of the REORG utility specifies the action that is to be taken if the REORG utility gets a timeout condition while trying to drain objects in either the log or switch phases. For additional information, refer to *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.

### 6.3.2 Deadlock avoidance between utilities and SQL

Deadlocks between utilities and SQL statements will not be detected by IRLM. Typically, an application gets a -911 (lock timeout) SQLCODE and utilities would end with RC=8 and reason code=00C200EA.

In order to avoid claim and drain deadlocks caused by the order in which SQL claims data and index objects, DB2 ensures that SQL jobs will always acquire claims on a table space or partition prior to acquiring claims on any of its indexes. Those utilities that drain multiple objects have also been modified to drain the data before the index(es).

To avoid claim and drain deadlocks caused by the order in which SQL claims partitions of a table space, the table space level claim is always acquired before any partition claim. Those utilities that operate on an entire partitioned table space will acquire the table space level drain before any partition drain.

## 6.4 Partition independence

Partition independence provides parallel processing support on the different partitions of a partitioned table space and its related indexes. Partition independence allows parallel execution of all utility jobs, SQL applications, and DB2 commands as long as they address different partitions of a given table space. This applies to partitioned tables spaces, universal partitioned-by-growth (PBG), and universal partitioned-by-range (PBR) table spaces.

Partition independence provides the following benefits:

- ▶ It is possible to independently access different partitions of a table space or index:
  - Different utilities can run on different partitions concurrently.
  - SQL applications run on partitions not in use by utilities.
  - It is possible to isolate unavailability of data to a single partition.
  - It is possible to start and stop a single partition.
- ▶ It is possible to independently access logical partitions of a non-partitioned index.
  - Different utilities can access the non-partitioned index concurrently.
  - SQL applications can access the logical partitions not in use by utilities.
  - It is possible to start and stop a logical partition.

A logical partition is the set of key and row identification (RID) pairs in a non-partitioned index that are associated with a particular partition. The RID contains the partition number to which the key entry belongs. Logical partitions are determined on this number.

Figure 6-2 shows an example of partition independence.

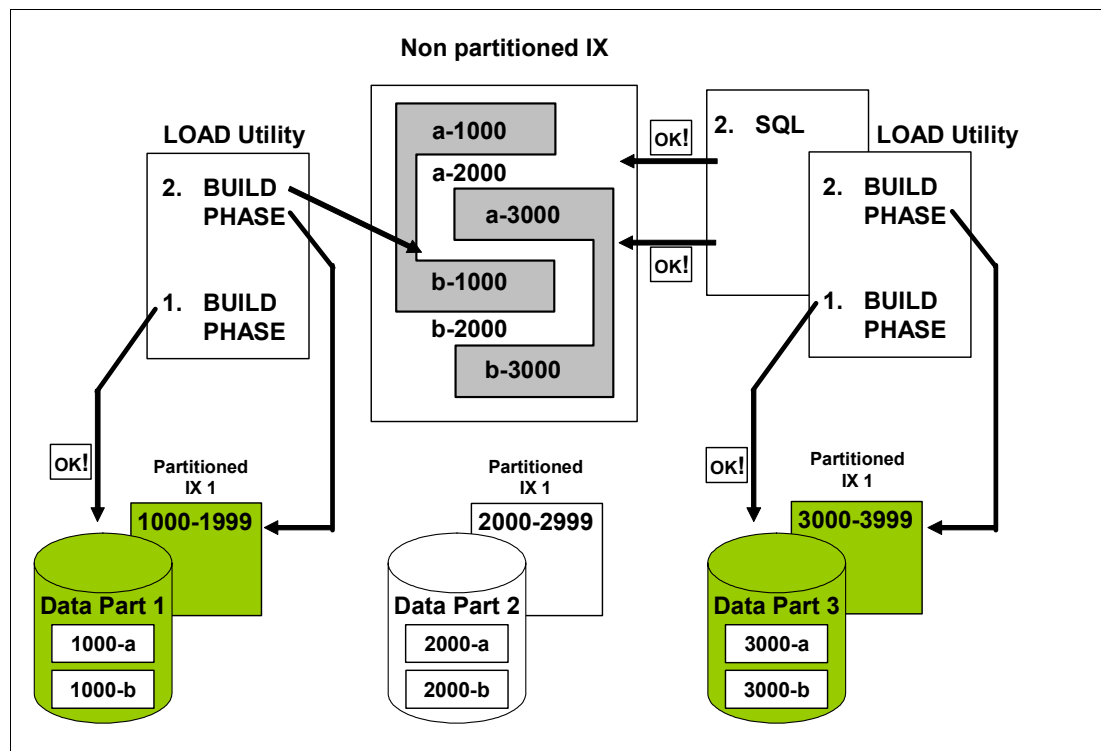


Figure 6-2 Example of partition independence

Figure 6-2 shows that the LOAD utility BUILD phases and SQL applications can operate concurrently on the different partitions, including the logical partitions of a non-partitioned index. A logical partition constitutes an independent entity that can be drained without impacting other logical partitions.

## 6.4.1 Scenario with LOAD PART and concurrent SQL updates

Figure 6-3 illustrates how SQL applications and DB2 utilities can operate concurrently on separate partitions of the same table space.

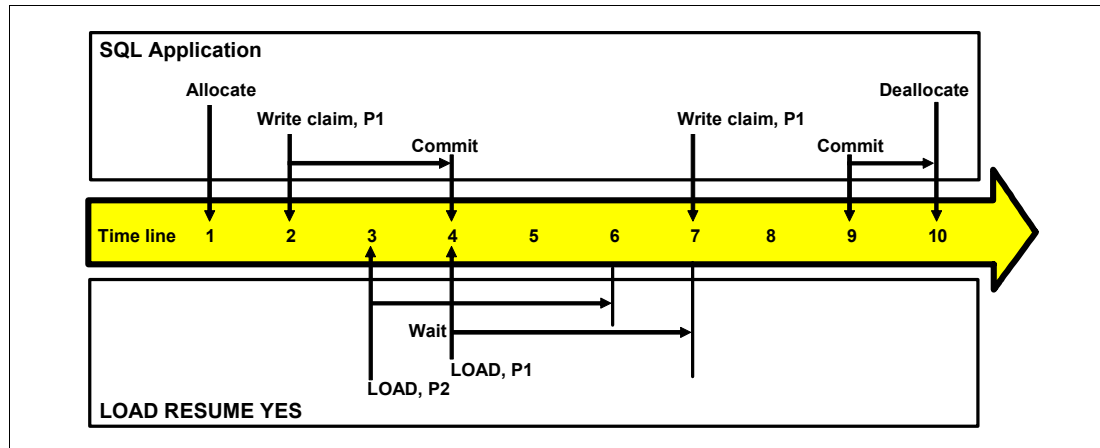


Figure 6-3 Two LOAD jobs execute concurrently with SQL on two partitions of a table space

For a partition-directed LOAD, if you specify SHRLEVEL CHANGE, only RESUME YES can be specified or inherited from the LOAD statement. LOAD SHRLEVEL CHANGE does not perform the SORT, BUILD, SORTBLD, INDEXVAL, ENFORCE, or REPORT phases. A regular LOAD job drains the entire table space, while a LOAD SHRLEVEL CHANGE functions like an INSERT statement and uses claims when accessing an object. Normally, a LOAD RESUME YES job loads the records at the end of the already existing records. However, for a LOAD RESUME YES job with the SHRLEVEL CHANGE option, the utility tries to insert the new records in available free space as close to the clustering order as possible. This LOAD job does not create any additional free pages.

LOAD RESUME SHRLEVEL CHANGE activates the before triggers and after triggers for each row that is loaded, so additional locks may be acquired, depending on the trigger action.

The scenario described in Figure 6-3 assumes that there are no NPIs on the table involved. Also, the secondary indexes, if any, are all data partitioned secondary indexes (DPSIs).

Table 6-15 explains the events associated with various points in the time line shown in Figure 6-3.

Table 6-15 Events associated with concurrently running LOAD utilities along with SQL

Time	Event
T1	An SQL application obtains a transaction lock on every partition in the table space. The duration of the locks extends until the table space is deallocated.
T2	The SQL application makes a write claim on data partition 1 and index partition 1.
T3	The LOAD jobs begin draining all claim classes on data partitions 1 and 2 and index partitions 1 and 2. LOAD on partition 2 operates concurrently with the SQL application on partition 1. LOAD on partition 1 waits.
T4	The SQL application commits, releasing its write claims on partition 1. LOAD on partition 1 can begin.
T6	LOAD on partition 2 completes.



Time	Event
T7	LOAD on partition 1 completes, releasing its drain locks. The SQL application (if it has not timed out) makes another write claim on data partition 1.
T9	The SQL application commits, releasing its write claims on partition 1.
T10	The SQL application deallocates the table space and releases its transaction locks.

**Consideration:** Applications that use repeatable read and access a non-partitioned index cannot run concurrently with utility operations that drain all claim classes of the non-partitioned index, even if they are accessing different logical partitions. For example, an application bound with ISOLATION(RR) cannot update partition 1 while the LOAD utility loads data into partition 2. Concurrency is restricted because the utility needs to drain all the repeatable-read applications from the non-partitioned index to protect the repeatability of the reads by the application.

## 6.4.2 Utility operations with non-partitioned indexes

In a non-partitioned index, either a partitioning index or a secondary index, an index entry can refer to any partition in the underlying table space.

DB2 can process a set of entries of a non-partitioned index that all refer to a single partition and achieve the same results as for a partition of a partitioned index. Such a set of entries is called a logical partition of the non-partitioned index.

Suppose that two LOAD jobs execute concurrently on different partitions of the same table space. When the jobs proceed to build a partitioned index, either a partitioning index or a secondary index, they operate on different partitions of the index and can operate concurrently. Concurrent operations on different partitions are possible because the index entries in an index partition refer only to data in the corresponding data partition for the table.

Utility processing can be more efficient with partitioned indexes because, with the correspondence of index partitions to data partitions, they promote partition-level independence. For example, the REORG utility with the PART option can run faster and with more concurrency when the indexes are partitioned. REORG rebuilds the parts for each partitioned index during the BUILD phase, which can increase parallel processing and reduce the lock contention of non partitioned indexes.

Similarly, for the LOAD PART and REBUILD INDEX PART utilities, the parts for each partitioned index can be built in parallel during the BUILD phase, which reduces lock contention and improves concurrency. The LOAD PART utility also processes partitioned indexes with append logic, instead of the insert logic that it uses to process non-partitioned indexes, which also improves performance.

DPSIs can promote partition independence, reduce lock contention, and improve index availability, especially for utility processing, partition-level operations (such as dropping or rotating partitions), and recovery of indexes.

**Tip:** Partition all your secondary indexes, if feasible.

## 6.5 Online utilities and concurrency

DB2 utilities with the SHRLEVEL CHANGE option are generally considered as online utilities. Some common online utilities that may be of interest to application developers is discussed in this section.

Utilities make use of claims and drains to control concurrency. Unlike the transaction locks, claims and drains are managed by DB2. But, the drain in turn acquires a drain lock, which is a regular lock managed by IRLM at the pageset level.

DB2 utilities with the SHRLEVEL CHANGE option generally do not acquire any transaction locks (except the LOAD and UNLOAD utilities), but may acquire drain locks in some phases of the execution. A few examples are illustrated in 6.5.3, “Online utilities that acquire drain lock” on page 177.

### 6.5.1 Transaction locks on online LOAD and UNLOAD utilities

The LOAD and UNLOAD utilities with the SHRLEVEL CHANGE option may acquire transaction locks at the page or row level.

#### **LOAD RESUME**

A LOAD SHRLEVEL CHANGE job functions like a mass INSERT. If the LOCKMAX or NUMLKTS value is not high enough, it could result in lock escalation and can severely restrict concurrent processing. Lock escalation will be disabled on XML table spaces for LOAD SHRLEVEL CHANGE.

Specifying LOAD RESUME (rather than PART integer RESUME) tells LOAD to serialize on the entire table space, which can inhibit concurrent processing of separate partitions. If you want to process other partitions concurrently, specify PART integer RESUME.

#### **UNLOAD with SHRLEVEL CHANGE ISOLATION UR**

Indicates that uncommitted rows, if they exist, are to be unloaded. The unload operation is performed with minimal interference from the other DB2 operations that are applied to the objects from which the data is being unloaded.

#### **UNLOAD with SHRLEVEL CHANGE ISOLATION CS**

Specifies that rows can be read, inserted, updated, and deleted from the table space or partition while the data is being unloaded.

The SKIP LOCKED DATA option specifies that the UNLOAD utility is to skip rows on which incompatible locks are held by other transactions. This applies to row level or page level lock.

### 6.5.2 UNLOAD from image copy data sets

When you specify FROMCOPY option, the UNLOAD utility processes only the specified image copy data set. Alternatively, you can use the FROMCOPYDDN keyword where multiple image copy data sets can be concatenated under a single DD name. You cannot unload LOB data or XML data from copies.

UNLOAD from DB2 image copy data sets claims the read claim class and concurrent read-write access is allowed. The table space (or physical partition) will be in a utility restrictive state to prevent the table space from being dropped while data is being unloaded from the image copy, even though the UNLOAD utility does not access the data in the table space.

No transaction locks are acquired on the base table.

If feasible, UNLOAD from image copy data sets should be used to attain maximum concurrency.

### 6.5.3 Online utilities that acquire drain lock

The following utilities with the SHRLEVEL CHANGE option acquire drain lock(s) and hence may limit the concurrency,

#### **REORG TABLESPACE (during the last iteration of LOG phase)**

REORG TABLESPACE consists of three phases:

- ▶ UNLOAD PHASE: Claims the read claim class, and concurrent SQL read-write access is allowed. The table space is in a utility restrictive state.
- ▶ Last iteration of LOG PHASE: Drains all the writers (default). Read-only access is allowed for concurrent DML processes. If the DRAIN ALL option is used, it will drain all the claim classes.
- ▶ SWITCH PHASE: Drains all claim classes. No concurrent SQL access is allowed.

#### **REORG INDEX**

REORG INDEX consists of three phases:

- ▶ UNLOAD PHASE: Claims the read claim class, and read-write access is allowed. The index is in a utility restrictive state.
- ▶ BUILD PHASE: No restrictions.
- ▶ Last iteration of LOG PHASE: Drains all the writers (default). Read-only access is allowed for concurrent DML processes. If the DRAIN ALL option is used, it will result in the draining of all claim classes.
- ▶ SWITCH PHASE: Drains all claim classes. No concurrent SQL access is allowed.

#### **Online REORG best practices for concurrency**

If you get timeouts or deadlocks when you use REORG with the SHRLEVEL CHANGE option, run the REORG utility with the DRAIN ALL option. The default is DRAIN WRITERS, which is done in the log phase. The specification of DRAIN ALL indicates that both writers and readers are drained when the MAXRO threshold is reached. Consider the DRAIN ALL option in environments where a lot of update activity occurs during the log phase. With this specification, no subsequent drain is required in the switch phase.

Consider scheduling the SWITCH phase in a maintenance window to avoid concurrent workloads that may prevent the utility from breaking in:

- ▶ MAXRO DEFER and LONGLOG CONTINUE will let REORG do its job except for the last log iteration and the switching.
- ▶ REORG will continue applying log until MAXRO is changed with the ALTER UTILITY command.

- ▶ Many log iterations might reduce the “perfect” organization of the table space, so keep the time until MAXRO is changed to keep final processing down to a minimum.

## 6.5.4 Online utilities without drain

The following utilities can be coded with the SHRLEVEL CHANGE option, which allows concurrent processing with SQL updates as well as reads and do not acquire any transaction or drain locks.

### **COPY SHRLEVEL CHANGE**

- ▶ Claims the read claim class.
- ▶ Read-write access allowed. This is a utility restrictive state.
- ▶ Does not allow concurrent mass deletes on segmented table spaces.

### **RUNSTATS TABLESPACE SHRLEVEL CHANGE**

- ▶ Claims the read claim class on the table space or partition.
- ▶ Allows other programs to change the table space. This is a utility restrictive state.
- ▶ The index is not at all affected by this utility.
- ▶ Does not allow concurrent mass deletes on segmented table spaces.
- ▶ May collect statistics on uncommitted data.

### **RUNSTATS INDEX SHRLEVEL CHANGE**

- ▶ Claims the read claim class on the index or partition.
- ▶ Allows other programs to change the index. This is utility restrictive state.
- ▶ Table space is not at all affected by this utility.
- ▶ May collect statistics on uncommitted data.

### **REBUILD INDEX SHRLEVEL CHANGE**

- ▶ Claims the read claim class.
- ▶ Read-write access is allowed. This is a utility restrictive state.
- ▶ Not suited for unique indexes with concurrent DML because the index is placed in RBDP while being built. Inserts, deletes and updates of the index will fail with a resource unavailable (-904) because uniqueness checking cannot be done while the index is in RBDP.
- ▶ REBUILD INDEX SHRLEVEL CHANGE jobs cannot be run to rebuild indexes on the same table space concurrently.
- ▶ Concurrency for rebuilding indexes in different table space is allowed, as is the concurrency in rebuilding different partitions of an index in a partitioned table space.

### **CHECK INDEX SHRLEVEL CHANGE**

- ▶ Drains all writers and forces the buffers to disk for the specified object and all of its indexes.
- ▶ Enables read-write access for the specified object and all of its indexes.

### **CHECK DATA SHRLEVEL CHANGE**

Applications can read from and write to the index, table space, or partition that is to be checked.

## 6.6 DB2 command considerations

The drain mechanism enables certain DB2 commands to break in on a thread holding claims on the object of interest even if those claims/locks were acquired by programs bound with `RELEASE(DEALLOCATE)`. When claims already held at the initiation of the drain process are released in the course of commit activity, the drain lock is obtained and the DB2 command can proceed with execution. In this section, we discuss `START`, `STOP`, and `BIND` commands, which are of some interest to the application developers. The interaction between the `BIND` command and different SQL processes is discussed in 6.7.1, “Interaction between bind, DDL, and DML” on page 191.

### 6.6.1 `START` command

The `START DATABASE` command makes the specified database available for use. Depending on which options you specify, the following objects can be made available for read-only processing, read-write processing, or utility-only processing:

- ▶ Databases
- ▶ Table spaces
- ▶ Index spaces
- ▶ Physical partitions of partitioned table spaces or index spaces (including index spaces housing data-partitioned secondary indexes (DPSIs))
- ▶ Logical partitions of non-partitioned secondary indexes

When a `START DATABASE` command for a restricted mode (RO and UT) takes effect depends on whether applications are started after the `START DATABASE` command has completed, or whether applications are executing at the time the command is issued. For applications that are started after `START DATABASE` has completed, access restrictions are effective immediately. For applications that are executing at the time `START DATABASE` is issued, the access restrictions take effect when a subsequent claim is requested or the application is allowed to run to completion.

Whether the application is interrupted by the `START DATABASE` command depends on various factors. These factors include the `ACCESS` mode that is specified on the `START DATABASE` command, the type of drain activity, if any, on the table space or partition, and whether any cursors are being held on the table space or partition. Refer to 6.8, “Restricted states” on page 193 for more information about how restricted states affect concurrency.

#### **`START ... ACCESS(RO)` and concurrent SQL reads**

DB2 attempts to acquire an S lock on table spaces (a gross lock) that are started with read-only access. If a partition is started with read-only access, DB2 attempts to get an S lock on the partition (a gross lock) that is started RO.

## 6.6.2 STOP command

The STOP DATABASE command makes the specified objects unavailable for applications and closes their data sets. Table 6-16 summarizes the locking used by the STOP DATABASE command.

Table 6-16 Locks and drains used by the STOP DATABASE command

Command	Table space type	Locks acquired
STOP	Partitioned	X lock partitions specified. Drain all on partitions specified.
		X lock all partitions. Drain all on all partitions (if PART is not specified).
	Non-partitioned	X lock table space. Drain all on table space.
STOP AT COMMIT	Partitioned	IX mass delete lock. Drain all on partitions specified.
		IX lock all partitions. Drain all on all partitions (if PART is not specified).
	Non-partitioned	IX mass delete lock. Drain all on table space.

## 6.6.3 BIND command

The following BIND options determine when an application process acquires and releases its locks and to what extent it isolates its actions from possible effects of other processes acting concurrently.

### ACQUIRE and RELEASE

The ACQUIRE and RELEASE options of BIND determine when DB2 locks an object (table, partition, or table space) your application uses and when it releases the lock. (The ACQUIRE and RELEASE options do not affect page, row, LOB, or XML locks.)

The options apply to static SQL statements, which are bound before your program executes. If your program executes dynamic SQL statements, the objects they lock are locked when first accessed and released at the next commit point, although some locks acquired for dynamic SQL might be held past commit points.

The ACQUIRE and RELEASE options are:

► ACQUIRE(ALLOCATE)

Acquires the lock when the object is allocated. This option is not allowed for BIND or REBIND PACKAGE.

► ACQUIRE(USE)

Acquires the lock when the object is first accessed.

► RELEASE(DEALLOCATE)

Releases the lock when the object is deallocated (the application ends). The value has no effect on dynamic SQL statements, which always use RELEASE(COMMIT), unless you are using dynamic statement caching. The value also has no effect on packages that are executed on a DB2 server through a DRDA connection with the client system.

► **RELEASE(COMMIT)**

Releases the lock at the next commit point, unless cursors are held. If the application accesses the object again, it must acquire the lock again.

The default options for ACQUIRE and RELEASE depend on the type of BIND option, as shown in Table 6-17.

*Table 6-17 Default ACQUIRE and RELEASE values for different bind options*

Operation	Default values
BIND PLAN	ACQUIRE(USE) and RELEASE(COMMIT).
BIND PACKAGE	No option exists for ACQUIRE; ACQUIRE(USE) is always used. At the local server, the default for RELEASE is the value used by the plan that includes the package in its package list. At a remote server, the default is COMMIT.
REBIND PLAN/PACKAGE	The existing value for the plan or package being rebound.

## Partition locks

Partition locks follow the same rules as table space locks, and all partitions are held for the same duration. Thus, if one package is using RELEASE(COMMIT) and another is using RELEASE(DEALLOCATE), all partitions use RELEASE(DEALLOCATE).

## Dynamic statement caching

Generally, the RELEASE option has no effect on dynamic SQL statements with one exception: When you use the bind options RELEASE(DEALLOCATE) and KEEP\_DYNAMIC(YES), and your subsystem is installed with YES for field CACHE\_DYNAMIC SQL on installation panel DSNTIP4, DB2 retains prepared SELECT, INSERT, UPDATE, and DELETE statements in memory past commit points. For this reason, DB2 can honor the RELEASE(DEALLOCATE) option for these dynamic statements. The locks are held until deallocation, or until the commit after the prepared statement is freed from memory, in the following situations:

- The application issues a PREPARE statement with the same statement identifier.
- The statement is removed from memory because it has not been used.
- An object that the statement is dependent on is dropped or altered, or a privilege needed by the statement is revoked.
- RUNSTATS is run against an object that the statement is dependent on.

If a lock is to be held past commit and it is an S, SIX, or X lock on a table space or a table in a segmented table space, DB2 sometimes demotes that lock to an intent lock (IX or IS) at commit. DB2 demotes a gross lock if it was acquired for one of the following reasons:

- DB2 acquired the gross lock because of lock escalation.
- The application issued a LOCK TABLE.
- The application issued a mass delete (DELETE FROM object without a WHERE clause or TRUNCATE).

Choose a combination of values for ACQUIRE and RELEASE based on the characteristics of the particular application.

### ***Example of dynamic statement caching***

An application selects employee names and telephone numbers from a table, according to different criteria. Employees can update their own telephone numbers. They can perform several searches in succession. The application is bound with the options ACQUIRE(USE) and RELEASE(DEALLOCATE), for these reasons:

1. The alternative to ACQUIRE(USE), ACQUIRE(ALLOCATE), gets a lock of mode IX on the table space as soon as the application starts, because that is needed if an update occurs. Most uses of the application do not update the table and so need only the less restrictive IS lock. ACQUIRE(USE) gets the IS lock when the table is first accessed, and DB2 promotes the lock to mode IX if that is needed later.
2. Most uses of this application do not update and do not commit. For those uses, little difference exists between RELEASE(COMMIT) and RELEASE(DEALLOCATE). However, administrators might update several phone numbers in one session with the application, and the application commits after each update. In that case, RELEASE(COMMIT) releases a lock that DB2 must acquire again immediately. RELEASE(DEALLOCATE) holds the lock until the application ends, avoiding the processing needed to release and acquire the lock several times.

Different combinations of BIND options have advantages and disadvantages for certain situations.

### ***ACQUIRE(ALLOCATE) / RELEASE(DEALLOCATE)***

In some cases, this combination can avoid deadlocks by locking all needed resources as soon as the program starts to run. This combination is most useful for a long-running application that runs for hours and accesses various tables, because it prevents an untimely deadlock from wasting that processing.

All tables or table spaces used in DBRMs bound directly to the plan are locked when the plan is allocated. (LOB and XML table spaces are not locked when the plan is allocated and are only locked when accessed.)

All tables or table spaces are unlocked only when the plan terminates.

The locks used are the most restrictive needed to execute all SQL statements in the plan regardless of whether the statements are actually executed.

Restrictive states are not checked until the page set is accessed. Locking when the plan is allocated ensures that the job is compatible with other SQL jobs. Waiting until the first access to check restrictive states provides greater availability; however, it is possible that an SQL transaction could:

- ▶ Hold a lock on a table space or partition that is stopped
- ▶ Acquire a lock on a table space or partition that is started for DB2 utility access only (ACCESS(UT))
- ▶ Acquire an exclusive lock (IX, X) on a table space or partition that is started for read access only (ACCESS(RO)), thus prohibiting access by readers

Disadvantages: This combination reduces concurrency. It can lock resources in high demand for longer than needed. Also, the option ACQUIRE(ALLOCATE) turns off selective partition locking; if you are accessing a partitioned table space, all partitions are locked.



This combination is not allowed for BIND PACKAGE. Use this combination if processing efficiency is more important than concurrency. It is a good choice for batch jobs that would release table and table space locks only to reacquire them almost immediately. It might even improve concurrency, by allowing batch jobs to finish sooner. Generally, do not use this combination if your application contains many SQL statements that are often not executed.

### ***ACQUIRE(USE) / RELEASE(DEALLOCATE)***

This combination results in the most efficient use of processing time in most cases. A table, partition, or table space used by the plan or package is locked only if it is needed while running. All tables or table spaces are unlocked only when the plan terminates.

The least restrictive lock needed to execute each SQL statement is used, with the exception that if a more restrictive lock remains from a previous statement, that lock is used without change.

Disadvantages: This combination can increase the frequency of deadlocks. Because all locks are acquired in a sequence that is predictable only in an actual run, more concurrent access delays might occur.

### ***ACQUIRE(USE) / RELEASE(COMMIT)***

This combination is the default combination and provides the greatest concurrency, but it requires more processing time if the application commits frequently.

A table, partition, or table space is locked only when needed. That locking is important if the process contains many SQL statements that are rarely used or statements that are intended to access data only in certain circumstances.

All tables and table spaces are unlocked when:

- ▶ In TSO, batch, and CAF: An SQL COMMIT or ROLLBACK statement is issued, or your application process terminates.
- ▶ In IMS: A CHKP or SYNC call (for single-mode transactions), a GU call to the I/O PCB, or a ROLL or ROLB call is completed.
- ▶ In CICS: A SYNCPOINT command is issued.

Exception: If the cursor is defined WITH HOLD, the table or table space locks necessary to maintain cursor position are held past the commit point.

Table, partition, or table space locks are released at the next commit point unless the cursor is defined WITH HOLD.

The least restrictive lock needed to execute each SQL statement is used except when a more restrictive lock remains from a previous statement. In that case, that lock is used without change.

Disadvantages: This combination can increase the frequency of deadlocks. Because all locks are acquired in a sequence that is predictable only in an actual run, more concurrent access delays might occur.

### ***ACQUIRE(ALLOCATE) / RELEASE(COMMIT)***

This combination is not allowed; it results in an error message from BIND.

## ABIND DSNZPARM

ABIND DSNZPARM specifies whether plans or packages are to be automatically rebound (or autobound) at run time in certain situations. Automatic rebinds can improve availability and administration of plans and packages because you do not need to explicitly rebind some plans and packages. However, automatic rebinds can also hinder performance because they require access to the DB2 directory and catalog.

ABIND DSNZPARM allows automatic rebind operations to be performed at run time when a plan or package is not valid; the SYSPLAN or SYSPACKAGE column VALID contains 'N'.

Changes to your program or database objects can invalidate plans and packages. Consider the effect of these changes when you determine the binding method for your program.

A change to your program probably invalidates one or more of your packages and perhaps your entire plan. For some changes, you must bind a new object; for others, rebinding is sufficient. A plan or package can also become invalid for reasons that do not depend on operations in your program. For example, when an index is dropped that is used in an access path by one of your queries, a plan or package can become invalid. In those cases, DB2 might rebind the plan or package automatically the next time that the plan or package is used.

Auto rebind happens in the following circumstances:

- ▶ Drop a table, index, or other object, and recreate the object.
- ▶ Drop an object that a package depends on.
- ▶ Revoke an authorization to use an object.

Rebind explicitly immediately after invalidation to avoid contentions in the online applications. For example, in a CICS environment, if multiple users try to access the same package (that was invalidated) at the same time, then multiple rebinds will be initiated simultaneously, resulting in error conditions.

## ISOLATION

The BIND command for application packages, as well as plans, have a parameter that controls the isolation level for all SQL statements executed by the program that is being bound. The parameter consists of the word ISOLATION followed by the abbreviation of the desired level, that is, RR, RS, CS, or UR.

The ISOLATION option of an application specifies the degree to which operations are isolated from the possible effects of other operations acting concurrently. Based on this information, DB2 releases S and U locks on rows or pages as soon as possible. Regardless of the isolation level that you specify, outstanding claims on DB2 objects can inhibit the execution of DB2 utilities or commands. The default ISOLATION option differs for different types of bind operations, as shown in Table 6-18.

*Table 6-18 The default ISOLATION values for different types of BIND operations*

Operation	Default values
BIND PLAN	ISOLATION (CS) with CURRENTDATA (NO)
BIND PACKAGE	Value used by the plan that includes the package in its package list
REBIND PLAN/PACKAGE	Existing value for the plan or package being rebound

The default isolation level, both for application programs and for interactive sessions, is Cursor Stability (CS).

**Note:** The ISOLATION value specified using the WITH clause in the SQL statement overrides the value specified for the ISOLATION option for the plan or package.

Choose an isolation level according to the needs and characteristics of the particular application. The four isolation levels are described in the order of increasing concurrency:

### **RR**

Repeatable Read is the highest level of isolation. It can prevent all the anomalies mentioned in 1.3, “The reasons for serialization” on page 5. A RR transaction with an index access acquires a lock on all data that it reads and holds the lock until the commit point. This prevents other transactions from updating/deleting/inserting any of this data. Thus, if a program reads the same piece of data twice in the same transaction with an isolation level of RR, it will see the same value (or absence of a value!). If an RR-level transaction reads a great amount of data, the concurrency of the database can be severely limited.

An index access prevents others from inserting a row, thus preventing the phantom anomaly. A RR transaction not using index access requires a gross lock on the table level.

### **RS**

Read Stability holds page or row locks until a COMMIT point, but other programs can INSERT new data. A row or page lock is held only for qualifying rows or pages that are returned to an application, at least until the next commit point. Figure 6-4 shows the locks held by program#1; a new row inserted by program#2 immediately after the row#4 was read by program#1.

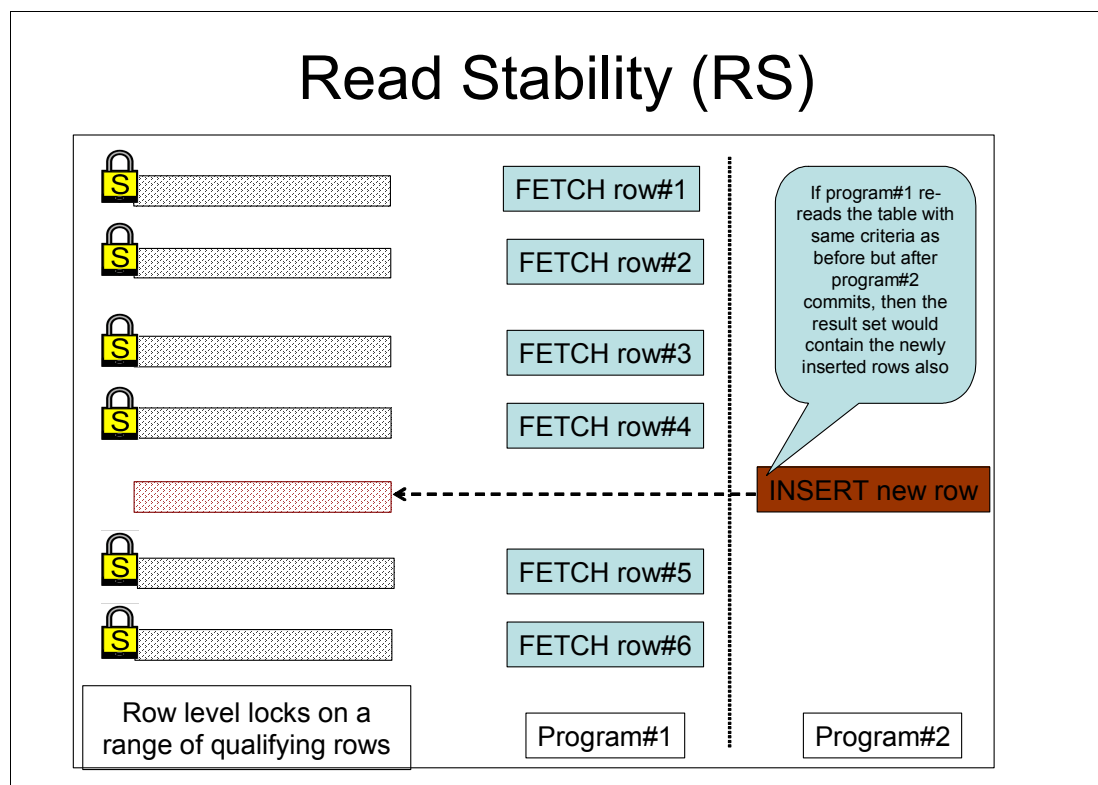


Figure 6-4 Potential phantom row anomaly during re-read with isolation RS

In this scenario, program#2 (any other program for that matter) cannot change rows that are returned to program#1 earlier. But, when program#1 tries to re-read the same rows after program#2 issues a commit, it would find an extra row.

This isolation level allows a transaction that reads the same row twice to receive the same value, but it does not prevent new rows from appearing during the course of a transaction. An RS-level transaction has a smaller impact on concurrency than an RR-level transaction, and it is protected from non-repeatable read anomalies but not from phantom row anomalies. The phantom row anomaly is also discussed in “Anomalies seen at various isolation levels” on page 9.

Applications using read stability can leave rows or pages locked for long periods, especially in a distributed environment. If you do use read stability, plan for frequent commit points to reduce the lock duration.

## **CS**

Cursor Stability acquires and releases page S locks as pages are read and processed. CS provides the greatest level of concurrency at the expense of potentially different data being returned by the same cursor if it is processed twice during the same unit of work.

If the cursor had specified FOR UPDATE OF, then DB2 would have taken U locks instead of S locks. U locks are also released as soon as the subsequent “fetch” causes DB2 to move from this page/row into another one.

CS-level transactions are protected against dirty reads and lost updates, but not against phantoms and non-repeatable reads. Obviously, the CS level of isolation provides less protection and has less impact on concurrency.

This isolation level is a very common setting for DB2 applications.

With ISOLATION(CS), locks are always avoided on singleton SELECTs. Locks are also avoided on non-qualifying rows of read-only and ambiguous cursors. But, lock avoidance on qualifying rows depends on the CURRENTDATA bind parameter setting.

For packages and plans that contain updateable static scrollable cursors, ISOLATION(CS) lets DB2 use optimistic concurrency control.

**Attention:** The RRULOCK DSNZPARM option determines the mode of lock chosen for a cursor defined with the FOR UPDATE OF clause and bound with read stability (RS) or repeatable read (RR). If you specify NO, the lock mode for operations with RR or RS is S (SHARE). If the cursor in your applications includes the FOR UPDATE OF clause, but updates are infrequent, S locks generally provide better performance. If you specify YES, the lock mode for operations with RR or RS is U. If your applications make frequent updates with repeatable-read isolation, the U lock might provide greater concurrency and perform better than that of the S lock. However, applications that require high concurrency are almost always more efficient if they use cursor stability (CS) isolation.

## **UR**

The ISOLATION (UR) or uncommitted read option allows an application to read while acquiring few locks, at the risk of reading uncommitted data. UR isolation applies only to read-only operations: SELECT, SELECT INTO, or FETCH from a read-only result table.

Uncommitted Read is also known as “dirty read”. UR avoids locking for isolation and provides the least amount of protection against isolation anomalies. A UR-level transaction has virtually no effect on concurrency. Since a UR-level transaction can read data that is in an inconsistent state (for example, it may see the debit to your savings account but not the credit to your checking account), this isolation level is usually used only in statistical surveys or other applications where 100% accuracy is not required.

The UR user has the least impact on the system. Any updaters in the system do not cause a lock wait for the UR user. The UR user does not cause a lock wait for any updaters.

The isolation (UR) level only applies to the “read” process. An “update” or “delete” or “insert” process will automatically use CS, if it is bound with UR and DB2 will not issue any warning/notification during BIND or run time when this happens.

## **CURRENTDATA**

The CURRENTDATA option tells whether the data upon which your cursor is positioned must remain identical to (or “current with”) the data in the local base table and applies only when ISOLATION(CS) is in effect. This option can be specified at a plan or package level, both in bind and rebind processes.

The ISOLATION (CS) or cursor stability option allows maximum concurrency with data integrity. However, after the process leaves a row or page, another process can change the data. With CURRENTDATA(NO), the process does not have to leave a row or page to allow another process to change the data. If the first process returns to read the same row or page, the data is not necessarily the same. Consider these consequences of that possibility:

For table spaces created with LOCKSIZE ROW, PAGE, or ANY, a change can occur even while executing a single SQL statement, if the statement reads the same row more than once. In the following statement, data read by the inner SELECT can be changed by another transaction before it is read by the outer SELECT:

```
SELECT * FROM T1
WHERE C1 = (SELECT MAX(C1) FROM T1);
```

Therefore, the information returned by this query might be from a row that is no longer the one with the maximum value for C1.

The CURRENTDATA option has different effects, depending on whether access is local or remote.

### ***CURRENTDATA for local access***

CURRENTDATA (YES) means that the data upon which the cursor is positioned cannot change while the cursor is positioned on it. If the cursor is positioned on data in a local base table or index, then the data returned with the cursor is current with the contents of that table or index. If the cursor is positioned on data in a work file, the data returned with the cursor is current only with the contents of the work file; it is not necessarily current with the contents of the underlying table or index.

Figure 6-5 shows how an application using CS isolation with CURRENTDATA(YES) acquires locks. This figure shows access to the base table. The L2 and L4 locks are released after DB2 moves to the next row or page. When the application commits, the last lock is released.

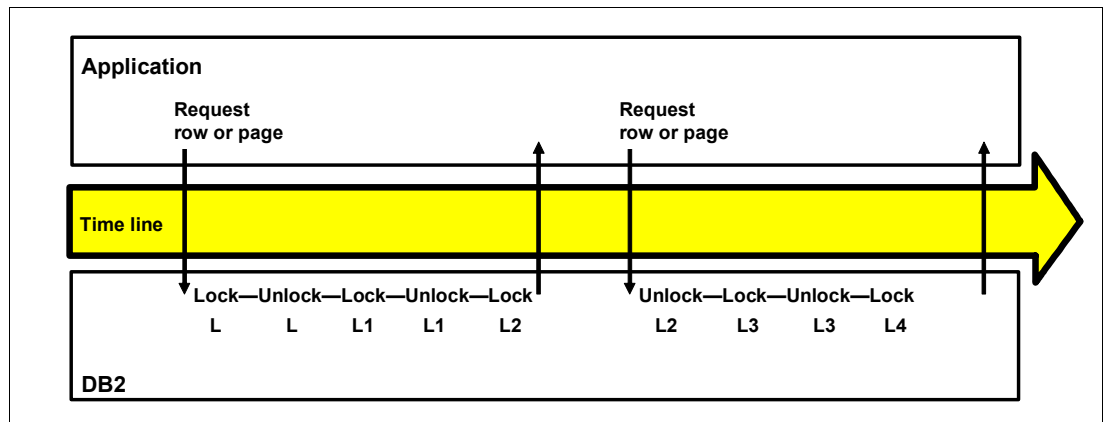


Figure 6-5 Locking with CURRENTDATA (YES)

With CURRENTDATA(NO), you have a much greater opportunity for avoiding locks. The following example shows how DB2 can avoid taking locks.

If the cursor appears to be used only for read-only, but dynamic SQL could modify data through the cursor, then the cursor is ambiguous. If you use CURRENTDATA to indicate an ambiguous cursor is read-only when it is actually targeted by dynamic SQL for modification, you will get an error. Ambiguous cursors can sometimes prevent DB2 from using lock avoidance techniques. However, misuse of an ambiguous cursor can cause your program to receive a -510 SQLCODE, which means:

- The plan or package is bound with CURRENTDATA(NO).
- An OPEN CURSOR statement is performed before a dynamic DELETE WHERE CURRENT OF statement against that cursor is prepared.
- One of the following conditions is true for the open cursor:
  - Lock avoidance is successfully used on that statement.
  - Query parallelism is used.
  - The cursor is distributed, and block fetching is used.

In all cases, it is a good programming technique to eliminate the ambiguity by declaring the cursor with either the FOR FETCH ONLY or the FOR UPDATE clause.

Figure 6-6 shows access to the base table. If DB2 must take a lock, then locks are released when DB2 moves to the next row or page, or when the application commits (the same as CURRENTDATA(YES)).

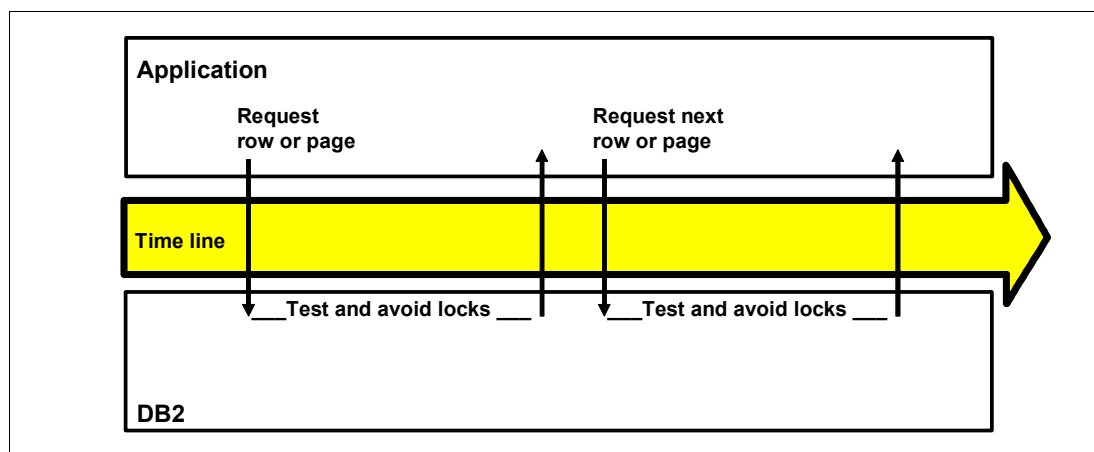


Figure 6-6 Best case of lock avoidance using isolation CS with CURRENTDATA(NO)

### **CURRENTDATA for remote access**

For a request to a remote system, CURRENTDATA has an effect on ambiguous cursors using isolation levels RR, RS, or CS. For access to a remote table or index, CURRENTDATA(YES) turns off block fetching for ambiguous cursors. The data returned with the cursor is current with the contents of the remote table or index for ambiguous cursors. Turning on block fetch offers the best performance, but it means the cursor is not current with the base table at the remote site.

The CURRENTDATA option has no effect in the following situations:

- ▶ For cursors positioned on data in a work file
- ▶ For read-only queries using parallelism
- ▶ For singleton SELECTs

In those cases, if you want to maintain currency with the data, you have the following options:

- ▶ Use isolation RR or RS (parallelism can still be used).
- ▶ Use the LOCK TABLE statement (parallelism can still be used).
- ▶ Disable parallelism (use SET DEGREE = '1' or bind with DEGREE(1)), if parallelism is the only reason for not maintaining currency and if the other two options are not feasible.

### **KEEPDYNAMIC**

The KEEPDYNAMIC option has locking implications for DRDA clients that specify WITH HOLD on their cursors.

If KEEPDYNAMIC(YES) is specified, the DB2 for z/OS server automatically closes the cursor when SQLCODE +100 is detected, which means that the client does not have to send a separate message to close the held cursor. Apart from reducing network traffic, it also reduces the duration of locks that are associated with the held cursor.

### **REOPT**

When the REOPT(ALWAYS) option is used on a static plan or package, DB2 determines the access path again at run time each time the statement is run. REOPT(VARS) is a synonym for REOPT(ALWAYS).

DB2 determines access paths at both bind time and run time for statements that contain one or more of the following variables:

- ▶ Host variables
- ▶ Parameter markers
- ▶ Special registers

At run time, DB2 uses the values in those variables to determine the access paths, just like dynamic SQL doing a full prepare, and therefore receives a DBD lock in S mode even when CACHEDYN=YES.

**Note:** If CACHEDYN=YES, and the statement qualifies for the cache, no lock is taken on the DBD when a statement is prepared for insertion in the cache or for a statement in the cache. If the statement does *not* qualify for the cache, DB2 acquires a DBD lock.

## 6.6.4 Conflicting BIND options

A plan bound with one set of options can include packages in its package list that were bound with different sets of options.

In general, statements in a DBRM bound as a package use the options with which the package was, and statements in DBRMs bound to a plan use the options with which the plan was bound. For example, the plan value for CURRENTDATA has no effect on the packages executing under that plan. If you do not specify a CURRENTDATA option explicitly when you bind a package, the default is CURRENTDATA(YES).

The rules are slightly different for the BIND options RELEASE and ISOLATION. The values of those two options are set when the lock on the resource is acquired and usually stay in effect until the lock is released. But a conflict can occur if a statement that is bound with one pair of values requests a lock on a resource that is already locked by a statement that is bound with a different pair of values. DB2 resolves the conflict by resetting each option with the available value that causes the lock to be held for the greatest duration.

If the conflict is between RELEASE(COMMIT) and RELEASE(DEALLOCATE), then the value used is RELEASE(DEALLOCATE).

If the isolation level of the plan and the package is different, DB2 uses the most restrictive isolation level in this sequence: RR > RS > CS > UR.

## 6.7 Locks on DB2 objects

There is some locking activity that is not related to user data, but takes place in shared DB2 subsystem objects. This locking activity can be caused by user application programs. It can also be caused by DB2 system plans processing. You are never aware of most of this locking activity, although in some cases it can produce suspension problems in your system.

The DB2 system plans that generate locking activity are:

- ▶ BCT, which is the plan used to perform service tasks and handle requests from DB2 resource managers. A lock issued by this plan is usually of short duration.
- ▶ ACT, which is the authorization plan used in the process of validating a user's authority to access a specified plan. A lock issued by this plan is of short duration, but ACT is used frequently (with every create thread).



- ▶ DSNBIND, which is the plan used for all binds. It is major contributor to the length of time it takes to complete a bind.
- ▶ DSNUTIL, which is the plan used by the DB2 utility control program DSNUTILB. The duration of locks held by this plan varies according to the specific DB2 utility function being invoked.
- ▶ DSNRRSAF, which is the plan used by Resource Recovery Services attachment facility (RRSAF), which enables programs to communicate with DB2.

### 6.7.1 Interaction between bind, DDL, and DML

DB2 uses different lock modes for different types of processes. The rows in Table 6-19 show a sample of several types of DB2 processes. The columns show the most restrictive mode of locks used for different objects and the possible conflicts between application processes due to locks on DB2 objects.

Table 6-19 Bind process interaction with SQL

Process	Catalog table spaces	Skeleton tables (SKCT and SKPT)	Database descriptor (DBD) <sup>a</sup>	Target table space <sup>b</sup>
BIND process	IX	X	S	n/a
Static Query (DML)	IS <sup>c</sup>	S	n/a <sup>d</sup>	Any <sup>e</sup>
Dynamic Query (DML)	IS <sup>f</sup>	S	S <sup>g</sup>	Any <sup>e</sup>
SQL CREATE TABLE statement	IX	n/a	X	n/a
SQL ALTER TABLE statement	IX	X <sup>h</sup>	X	n/a
SQL ALTER TABLESPACE statement	IX	X <sup>i</sup>	X	n/a
SQL DROP TABLESPACE statement	IX	X <sup>j</sup>	X	n/a
SQL GRANT statement	IX	n/a	n/a	n/a
SQL REVOKE statement	IX	X <sup>j</sup>	n/a	n/a

a. In a lock trace, these locks usually appear as locks on the DBD.

b. The target table space is one of the following table spaces:

- Accessed and locked by an application process
- Processed by a utility
- Designated in the data definition statement

c. The lock is held briefly to check EXECUTE authority.

d. If the required DBD is not already in the EDM DBD cache, locks are acquired on table space DBD01, which effectively locks the DBD.

e. See Table 5-1 on page 89, Table 5-2 on page 97, and Table 5-6 on page 105 for details about target table space locks requested by different types of DML statements.

f. Except while checking EXECUTE authority, IS locks on catalog tables are held until a commit point.

g. If caching of dynamic SQL is turned on (CACHEDYN=YES), no lock is taken on the DBD when a statement is prepared for insertion in the cache or for a statement in the cache.

h. The plan or package using the SKCT or SKPT is marked invalid if a referential constraint (such as a new primary key or foreign key) is added or changed, or the AUDIT attribute is added or changed for a table.

i. These locks are not held when ALTER TABLESPACE is changing the following options: PRIQTY, SECQTY, PCTFREE, FREEPAGE, CLOSE, and ERASE.

j. The plan or package using the SKCT or SKPT is marked invalid as a result of this operation.

## 6.7.2 Contention on the DB2 catalog

SQL data definition statements, GRANT statements, and REVOKE statements require locks on the DB2 catalog. If different application processes are issuing these types of statements, catalog contention can occur.

### Contention within the SYSDBASE table space

SQL statements that update the catalog table space SYSDBASE contend with each other when those statements are on the same table space. Those statements are:

- ▶ CREATE TABLESPACE, TABLE, and INDEX
- ▶ ALTER TABLESPACE, TABLE, INDEX
- ▶ DROP TABLESPACE, TABLE, and INDEX
- ▶ CREATE VIEW, SYNONYM, and ALIAS
- ▶ DROP VIEW and SYNONYM, and ALIAS
- ▶ COMMENT ON and LABEL ON
- ▶ GRANT and REVOKE of table privileges
- ▶ RENAME TABLE
- ▶ RENAME INDEX
- ▶ ALTER VIEW

**Recommendation:** Reduce the concurrent use of statements that update SYSDBASE for the same table space. When you alter a table or table space, quiesce other work on that object.

### Contentions independent of databases

The following limitations on concurrency are independent of the referenced database:

- ▶ CREATE and DROP statements for a table space or index that uses a storage group contend significantly with other such statements.
- ▶ CREATE, ALTER, and DROP DATABASE, and GRANT and REVOKE database privileges all contend with each other and with any other function that requires a database privilege.
- ▶ CREATE, ALTER, and DROP STOGROUP contend with any SQL statements that refer to a storage group and with extensions to table spaces and indexes that use a storage group.
- ▶ GRANT and REVOKE for plan, package, system, or use privileges contend with other GRANT and REVOKE statements for the same type of privilege and with data definition statements that require the same type of privilege.

## 6.7.3 Locks on skeleton tables

The skeleton table of a plan (SKCT) or package (SKPT) is locked while the plan or package is running.

The following operations require incompatible locks on the SKCT or SKPT, whichever is applicable, and cannot run concurrently:

- ▶ Binding, rebinding, or freeing the plan or package
- ▶ Dropping a resource or revoking a privilege that the plan or package depends on
- ▶ In some cases, altering a resource that the plan or package depends on

## 6.7.4 Locks on the database descriptors (DBDs)

Whether a process locks a target DBD depends largely on whether the DBD is already in the EDM DBD cache.

### If the DBD is not in the EDM DBD cache

Most processes acquire locks on the database descriptor table space (DBD01). That has the effect of locking the DBD and can cause conflict with other processes.

### If the DBD is in the EDM DBD cache

The lock on the DBD depends on the type of process, as shown in Table 6-20.

Table 6-20 Contention for locks on a DBD in the EDM DBD cache

Process Type	Process	Lock acquired	Conflicts with process type
1	Static DML statements (SELECT, DELETE, INSERT, or UPDATE) <sup>a</sup>	None	None
2	Dynamic DML statements <sup>b</sup>	S	3
3	Data definition statements (ALTER, CREATE, or DROP)	X	2, 3, or 4
4	Utilities	S	3

a. Static DML statements can conflict with other processes because of locks on data.

b. If caching of dynamic SQL is turned on, no lock is taken on the DBD when a statement is prepared for insertion in the cache or for a statement in the cache.

## 6.8 Restricted states

To control access and help ensure data integrity, DB2 sets a restrictive or nonrestrictive (advisory) status on certain objects. This topic outlines the restrictive and nonrestrictive (advisory) object statuses that affect utilities. For a full description of each status and the required steps to correct each status for a particular object, refer to the *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.

Use the DISPLAY DATABASE command to display the current status for an object.

Restricted states can be set on DB2 objects:

- ▶ Database
- ▶ Table space
- ▶ Partitioned table space
- ▶ Index space
- ▶ Partitioned index space

These states can be set by either a DB2 utility or a DB2 command. Restricted states are managed by DB2 with no involvement of IRLM.

Resetting a restricted state on a DB2 object can only be done by executing:

- ▶ A new DB2 utility against the DB2 object
- ▶ START DATABASE(database) SPACENAM(table space or index space) ACCESS(FORCE)
- ▶ Using the REPAIR utility against the DB2 object

Only the CHKP, COPY, PSRCP, and RECP restricted states can be reset.

If an application process requests a transaction lock on a table space that is in a restrictive status (for example, RECP) or has a required index in a restrictive status, DB2 acquires the lock. DB2 does not detect the status until the application tries to access the table space or index, when the application receives an error message indicating that the resource is not available (SQLCODE -904). After receiving this message, the application should release the lock, either by committing or rolling back (if the value of the RELEASE option is COMMIT) or by ending (if the value of RELEASE is DEALLOCATE). If you issue the command START DATABASE ACCESS(FORCE) for either the table space or the index space while the lock is in effect, the command fails.

Table 6-21 shows all the possible restricted states, how they are set, and how they can be reset on the DB2 objects. Table 6-22 on page 199 shows what is checked.

Table 6-21 Restricted states

Object state	Description	Objects affected	State set by	State reset by
ACHKP	The auxiliary CHECK-pending (ACHKP) restrictive status is set on when at least one base table LOB column error is detected and not invalidated as a result of running CHECK DATA AUXERROR REPORT.	Base table space.	RECOVER to point in time on the base table space without recovery of the LOB table spaces.	CHECK DATA AUXERROR REPORT.
AREO*	The advisory reorganize state indicates that the table space, index, or partition identified should be reorganized for optimal performance.	Table space, partition, or Index space.	<ul style="list-style-type: none"> <li>► Alter table column and alter index column in same commit scope.</li> <li>► Alter add identity column.</li> <li>► Alter add row change time stamp column.</li> </ul>	REORG TABLESPACE, REORG INDEX, LOAD REPLACE, REBUILD INDEX.
AREST	Indicates that an object is in advisory RESTART-pending state. If backout activity against the object is not underway, initiate it either by issuing the RECOVER POSTPONED command or by recycling the system with the system parameter LBACKOUT=AUTO.	Table space, index space, or a physical partition of a table space or index space.	DB2 sub-system start.	The AREST status is removed when the backout processing is complete for the object or RECOVER POSTPONED CANCEL is issued.
AUXW	Either the base table space or LOB table space is in the auxiliary warning advisory status, indicating an error in the LOB column, or the LOB table space is in the auxiliary warning advisory status, indicating an invalid LOB. This message can also indicate an error in a base table space with an XML column or in an XML table space.	Base table space. LOB table space. XML table space.	<ul style="list-style-type: none"> <li>► CHECK LOB.</li> <li>► RECOVER logapply.</li> </ul>	CHECK LOB, CHECK DATA.

Object state	Description	Objects affected	State set by	State reset by
CHKP	The CHECK-pending restrictive status indicates that an object might be in an inconsistent state and must be checked.	Table space, base table space, partitioning index, non partitioning index, index on the auxiliary table, and LOB table space.	<ul style="list-style-type: none"> <li>▶ LOAD ENFORCE NO.</li> <li>▶ LOAD REPLACE parent table.</li> <li>▶ ALTER TABLE ADD FOREIGN KEY.</li> <li>▶ Set by RECOVER to a point in time when the entire RI set is not included.</li> </ul>	CHECK DATA.
COPY	The COPY-pending restrictive status indicates that the affected object must be copied. The COPY pending state allows read access, but not write access.	Table space and table space partition.	LOAD, REORG LOG NO; MODIFY RECOVERY which removes all image copy info.	COPY FULL YES.
DBET	The DBET error status identifies the objects that need special attention by the user.	Table space, table space partition, index space, index partition, or logical index partition.	The error occurred when the DBET states were being modified during log apply or must-complete processing. This prohibited DB2 from successfully updating the DBET states.	Contact IBM Software Support to report the problem. The DB2 log records need to be analyzed to diagnose the cause of the problem and determine further actions.
GRECP	The group buffer pool RECOVER-pending status is set on when a coupling facility fails with pages that were not externalized. The affected object must be recovered.	All data objects.	Buffer Manager.	Recover the object, or use START DATABASE to recover the object.
ICOPY	The informational COPY-pending advisory status indicates that the affected object should be copied.	NOT LOGGED table spaces. Partitioning index, non-partitioning index, and index on the auxiliary table.	<ul style="list-style-type: none"> <li>▶ SQL INSERT, UPDATE, DELETE to a table in a NOT LOGGED tablespace.</li> <li>▶ LOAD REPLACE, REORG TABLESPACE, REORG INDEX, REBUILD INDEX.</li> </ul>	COPY FULL YES.
LPL	The object has entries in the logical page list.	Table space, table space partition, index, index partition, and logical index partition.	SQL.	START DATABASE, RECOVER, LOAD REPLACE, and SQL DROP.

Object state	Description	Objects affected	State set by	State reset by
LSTOP	The logical partition of a non partitioning index is stopped.	Logical partition of a non-partitioning index.	STOP DB command.	START DB command.
RBDP	A REBUILD-pending restrictive status indicates that the affected physical or logical index or index partition is broken and must be rebuilt from the data.	Physical or logical index partition.	<ul style="list-style-type: none"> <li>▶ Alter index padded to not padded when varying the key column length.</li> <li>▶ Alter the table add column and alter the index add column in the same commit scope.</li> <li>▶ Alter index compress yes   no.</li> <li>▶ Alter index piecesize.</li> <li>▶ Alter index bufferpool.</li> <li>▶ Create index defer yes.</li> <li>▶ RESTORE SYSTEM.</li> <li>▶ RECOVER to a point in time on the underlying table space.</li> <li>▶ Other utilities, such as REORG or LOAD, will set this state in case of an abend then -TERM UTIL.</li> </ul>	REBUILD INDEX, REORG TABLESPACE, and LOAD REPLACE.
RBDP*	The REBUILD-pending star restrictive status indicates that the logical partition of a non-partitioning index is in the REBUILD-pending status, and the entire index is inaccessible to SQL applications. However, only the logical partition needs to be rebuilt.	Logical partitions of non partitioned secondary indexes.	RECOVER to a point in time on the underlying table space partition.	REBUILD INDEX and LOAD REPLACE.
PSRBD	A page set REBUILD-pending restrictive status indicates that the affected index or index partition is broken and must be rebuilt from the data.	Non-partitioned secondary index and index on the auxiliary table.	As for RBDP.	As for RBDP.
PSRCP	A page set RECOVER-pending restrictive status of an index space that indicates that the entire page set must be recovered.	Non-partitioning index and index on the auxiliary table.	I/O error or utility termination.	Recover or rebuild the index.

Object state	Description	Objects affected	State set by	State reset by
RECP	The RECOVER-pending restrictive status indicates that a table space or table space partition is broken and must be recovered.	Table space. Table space partition Index. Index space. Index on the auxiliary table. Logical index partition.	RECOVER TS index error and other utilities, such as REORG or LOAD, will set this state in case of an abend and then set -TERM UTIL.	RECOVER and LOAD REPLACE.
REFP	Indicates that the object is in the REFRESH-pending status.	Table space, index space, or an index	-CANCEL THREAD when backout processing fails or when NOBACKOUT is specified.	REBUILD, LOAD REPLACE. RECOVER to a point in time.
RELDP	The object has a release dependency.	Any object.	Fall back to previous version.	Migration.
REORP	The REORG-pending restrictive status indicates that a table space partition definition has changed and the affected partitions must be reorganized before the data is accessible.	Table space and partitioned table space	<ul style="list-style-type: none"> <li>▶ Alter limit key of partitions.</li> <li>▶ Alter add identify column.</li> <li>▶ RECOVER to point in time prior to REORG that materialized #1 or #2.</li> </ul>	REORG TABLESPACE.
RESTP	The restart-pending status is set on if an object has backout work pending at the end of DB2 restart.	Table space, table space partitions, index spaces, and physical index space partitions.	DB2 system start; START DATABASE.	RECOVER POSTPONED command.
RO	The database, table space, table space partition, index space, or index space partition is started for read-only activity.			
RW	The database, table space, table space partition, index space, or index space partition is started for read and write activity.			
STOP	The database, table space, table space partition, index space, or index space partition is stopped.			
STOPE	The table space or index space was implicitly stopped because there is a problem with the log RBA in a page. Message DSNT500I or DSNT501I is issued when the error is detected, indicating the inconsistency.	Table space, table space partition, index, and index partition.	DB2 error.	Correct error.

Object state	Description	Objects affected	State set by	State reset by
STOPP	A stop is pending for the database, table space, table space partition, index space, or index space partition.	Table space, table space partition, index, index partition.	STOP DATABASE.	Release Claims.
UT	The database, table space, table space partition, index space, or index space partition is started for utility processing only.			
UTRO	A utility is in process, on the table space, table space partition, index space, or index space partition, that allows only RO access. If the utility was canceled before the object was drained, the object can allow SQL access because the object was not altered by the utility.			
UTRW	A utility is in process, on the table space, table space partition, index space, or index space partition, that allows RW access.			
UTUT	A utility is in process, on the table space, table space partition, index space, or index space partition, that allows only UT access. If the utility was canceled before the object was drained, the object can allow SQL access because the object was not altered by the utility.			
WEPR	Displays write error page range information.	Table spaces, index spaces, or partitions.	Buffer manager when a write I/O error is detected.	RECOVER or RECOVER with ERROR RANGE.

During execution of a program, DB2 checks if any restricted states exist on the referenced DB2 objects.



DB2 has grouped the restricted states into two different levels of checking:

- ▶ Level 1
  - STOP
  - START RO
  - START UT
  - CHKP
- ▶ Level 2
  - UTUT
  - UTRO
  - UTRW
  - COPY
  - PSRCP
  - RECP
  - CHKP
  - STOP AT (COMMIT)

Level 1 restricted states are always checked, while Level 2 restricted states are checked only when an application program makes a claim.

Table 6-22 shows when the restricted state checks are done during SQL program processing.

*Table 6-22 Restricted state check during SQL program process*

State	ACQUIRE(ALLOCATE) RELEASE(DEALLOCATE)	ACQUIRE(USE) RELEASE(COMMIT)	ACQUIRE(USE) RELEASE(DEALLOCATE)
Level 1 Restricted states	First reference and once per execution	First reference and for each unit of recovery	First reference and once per execution
Level 2 Restricted states	First reference and each unit of recovery	First reference and for each unit of recovery	First reference and for each unit of recovery

ACQUIRE(ALLOCATE) is possible only if data base request modules (DBRM) are directly bound to a plan. Packages are always bound with ACQUIRE(USE). For a plan bound with ACQUIRE(ALLOCATE), all table space locks that are needed in the program are acquired after the thread is created. Until the object is actually used, no claims are required. In this case, a command or utility can set or hold a restricted state on an object not yet referenced in the application program.

If the plan or package is bound with RELEASE (DEALLOCATE), table space locks are not released at commit point. The claim on the object is lost and a command or a utility can set a restricted state on this object.





## System considerations

In this chapter, we review the system parameter (DSNZPARMs) settings that could impact DB2 resource locking, concurrency, and availability. We also describe how to influence locking behavior and how to reduce locking overhead at the system level.

The following topics are covered:

- ▶ Application and system interaction
- ▶ IRLM start procedure parameters
- ▶ DB2 system parameters that directly affect locking

## 7.1 Application and system interaction

In this chapter, we describe the options for controlling the behavior of the IRLM and DB2 address spaces by way of the start parameters that are specified in the IRLM start procedure and the DB2 DSNZPARM. These parameters are set as part of the installation process and are documented in the *DB2 Version 9.1 for z/OS Installation Guide*, GC18-9846.

## 7.2 IRLM start procedure parameters

You can control how DB2 uses locks by specifying certain options when you start the Internal Resource Lock Manager (IRLM).

When the z/OS START IRLMPROC command is issued (normally automatically as part of starting a DB2 sub-system), the values of the options are passed to the startup procedure for the DB2 IRLM. If an option is not explicitly specified on the command, the value of its corresponding installation parameter is used.

The options that are relevant to DB2 locking are shown in Table 7-1.

Table 7-1 IRLM startup procedure options

Options	Description	Default
SCOPE	Use LOCAL if you are in a non data sharing environment. If you use data sharing, specify GLOBAL.	
DEADLOK <sup>a</sup>	The two values of this option specify: 1. The number of seconds between two successive scans for a local deadlock detection. 2. The number of local scans that occur before a scan for global deadlock starts. (This number is used only for DB2 data sharing.)	1 <sup>b</sup> ,1 <sup>c</sup>
PC	Ignored by IRLM. However, PC is positional and must be maintained in the IRLM for compatibility.	
MAXCSA	Ignored by IRLM. However, MAXCSA is positional and must be maintained in the IRLM for compatibility.	
MLMT	The MLMT parameter specifies the amount of storage available to IRLM “above the bar”, that is, the z/OS <i>memlimit</i> setting for the IRLM address space. IRLM uses 2G if you specify a value of less than 2G for MLMT.	2G

a. A deadlock is a situation where two or more requesters are waiting for resources that are held by another requester. Deadlock detection is the procedure by which a deadlock and its participants are identified.

b. DB2 interprets values between 1 and 5 as seconds and values between 100 and 5000 as milliseconds. Depending on the value that you enter, IRLM might substitute a smaller maximum value.

c. 1 is the only permissible value; any other value is ignored.

These values are set as part of the DB2 installation process. Thereafter, they can only be changed by editing the IRLM address space procedure or using an IRLM modify command.

The maximum amount of storage available for IRLM locks is limited to 90% of the total space given to the IRLM private address space during the startup procedure. The other 10% is reserved for IRLM system services, z/OS system services, and “must complete” processes to prevent the IRLM address space from abending, which would bring down your DB2 system.

When the storage limit is reached, lock requests are rejected with an out-of-storage reason code.

Use the MODIFY irlmproc,STATUS,STOR command to view and monitor the amount of private storage that IRLM has available, as shown in Example 7-1.

*Example 7-1 How to display the IRLM storage above the bar*

---

```
/MODIFY DB9AIRLM,STATUS,STOR
```

```
RESPONSE=SC63
DXR100I ID9A001 STOR STATS
PC: YES  LTEW:n/a  LTE:      M RLE:      RLEUSE:
BB PVT: 1310M  AB PVT (MEMLIMIT): 2048M
CSA USE: ACNT:      OK  AHWM:      OK  CUR: 926K  HWM: 926K
        ABOVE 16M: 26 926K  BELOW 16M: 0  OK
CLASS  TYPE  SEGS    MEM  TYPE  SEGS    MEM  TYPE  SEGS    MEM
ACCNT  T-1    1    2048K  T-2    1    1024K  T-3    1     4K
PROC   WRK    4     20K   SRB    1     1K   OTH    1     1K
MISC   VAR   19   5987K  N-V   12    316K  FIX    1    24K
DXR100I End of display
```

---

The description of the fields in message DXR100I can be found in the manual *IRLM Messages and Codes for IMS and DB2 for z/OS*, GC19-2666.

You can adjust the amount of below the bar private storage dynamically with the MODIFY irlmproc SET,PVT command. The IRLM command MODIFY irlmproc,SET,MLT can be used to dynamically change the maximum amount of IRLM private storage above the bar to use for locks, as shown in Example 7-2.

*Example 7-2 How to dynamically adjust the limit for above the bar private storage*

---

```
/MODIFY DB9AIRLM,SET,MLT=3G
```

```
DXR177I ID9A001 THE VALUE FOR MLT      IS SET TO      3G
```

---

The new value remains in effect until the next time IRLM is stopped and restarted or until the MODIFY command is issued successfully again. For below the bar private storage, this only changes the monitoring threshold of private storage for IRLM. For above the bar private storage, this only updates the MEMLIMIT that z/OS uses to control the amount of above the bar storage that can be requested by an address space. Neither command changes the physical amount that the operating system assigned to the address space.

Enabling data sharing further increases the storage that IRLM requires.

Sysplex query parallelism requires additional storage beyond what is required for data sharing.

## 7.2.1 Estimating the storage needed for locks

DB2 locks require storage space. That value is calculated during the DB2 installation process.

To estimate the storage that is required for DB2 locks:

1. For a conservative starting figure, assume that:
  - DB2 locks require 540 bytes of storage for each lock.
  - All concurrent threads hold the maximum number of row or page locks (LOCKS PER USER on installation panel DSNTIPJ). The number of table and table space locks is negligible.
  - The maximum number of concurrent threads that are active (MAX USERS + MAX REMOTE ACTIVE).
2. Then the storage can be calculated as:

$$\text{Storage} = 540 \times (\text{LOCKS PER USER}) \times (\text{MAX USERS} + \text{MAX REMOTE ACTIVE})$$

IRLM V2R2, the required level of IRLM to support DB2 9 for z/OS, is enabled for 64-bit processing and so is able to store the locks above the bar. The default, and minimum value, of 2 GB (2G) for the memory limit parameter (MLMT) is equivalent to approximately 3.7 million locks. The maximum number of locks is theoretically 100 million.

## 7.3 DB2 system parameters that directly affect locking

The locking options can be specified in the DSNZPARM DSN6SPRM macro, in the IRLM startup procedure, or during DB2 installation on the two IRLM panels DSNTIPI and DSNTIPJ. The major system parameters related to application locking and lock performance of the DB2 subsystem are discussed in this section.

Note that the DSNZPARMs apply to an entire DB2 subsystem, not just to one application program or one table.

### 7.3.1 IRLMRWT: RESOURCE TIMEOUT field

This is the entry RESOURCE TIMEOUT on the DB2 installation panel DSNTIPI and corresponds to the DSNZPARM IRLMRWT. It specifies the number of seconds IRLM waits before detecting a timeout. Acceptable values are 1 to 3600 seconds. The default is 60 seconds.

A timeout is a conditional lock request that has waited for a resource longer than the number of seconds specified for this option.

IRLM checks for timeouts on each deadlock detection cycle. So the actual wait time between the lock request and IRLM detecting the timeout will be:

$$\text{IRLMRWT} \leq \text{actual wait time} \leq \text{IRLMRWT} + \text{DEADLOK}$$

DEADLOK is the deadlock detection cycle time. So, for example, if the timeout is set to 15 seconds and the dead lock detection cycle is 2 seconds, then the actual wait time will be between 15 and 17 seconds.

Consider how long your system can afford to wait for a suspended process. If you can allow a suspended process to remain inactive for 60 seconds, use the default. A small value may cause a large number of timeouts. With a value that is too large, suspended processes more often resume normally, but they remain inactive for longer periods, which, if they are holding locks, can lead to more deadlocks and timeouts.

IRLMRWT DSNZPARM cannot be updated online, but you can use the operator /MODIFY,irlmproc to change the timeout value at the IRLM level. We also recommend that you consider using different values for online and batch (higher for the batch) applications, if feasible.

Decrease this value from the default value of 60 seconds, if you would like to clear out deadlocks or timeouts much faster, especially when using the latest IBM System z processors, such as the z10. Refer to 7.3.2, “DEADLOK (part 1): DEADLOCK TIME” on page 207 for a discussion about the deadlock parameters.

Example 7-3 shows how to display the current values for TIMEOUT and DEADLOK using a MODIFY command.

*Example 7-3 How to display the deadlock and timeout value*

---

```

/MODIFY DB9AIRLM,STATUS
RESPONSE=SC63
DXR101I ID9A001 STATUS SCOPE=LOCAL
      DEADLOCK: 1000
SUBSYSTEMS IDENTIFIED
NAME      T/OUT  STATUS      UNITS      HELD      WAITING      RET_LKS
DB9A      0060   UP           5           14          0            0
DXR101I End of display

```

---

Example 7-4 shows how to modify the TIMEOUT value using an IRLM modify command.

*Example 7-4 How to modify the timeout value at IRLM level*

---

```

/MODIFY DB9AIRLM,SET,TIMEOUT=200,DB9A

```

---

```

13.01.11 STC08084 DXR177I ID9A001 THE VALUE FOR TIMEOUT IS SET TO 200 FOR DB9A

```

---

Note that the timeout will revert back to the DB2 DSNZPARM setting when DB2 is next restarted.

Other processes have their timeout value set as a multiple of the IRLMRWT value, as shown in Table 7-2.

Table 7-2 Timeout multiplier and execution environment

Program execution environment	Multiple of IRLMRWT <sup>a</sup>	Modifiable?
IMS MPP, IMS Fast Path message processing, CICS, QMF, CAF, TSO batch and online, RRSAP, and global transactions.	1	No
IMS BMP	4	Yes with BMPTOUT DSNZPARM
IMS DL/I Batch	6	Yes, with DLITOUT DSNZPARM
IMS Fast Path non-message processing	6	No
BIND processing	3	No
STOP DATABASE command processing	10	No
DB2 Utilities	6	Yes, with UTIMOUT DSNZPARM
Retained locks when data sharing	0	Yes, with RETLWAIT DSNZPARM
Anything else	1	No

a. If the transaction occurs on a table space that is not logged, the timeout multiplier is either 3 or the current timeout multiplier for the thread, whichever is greater.

Setting the timeout value is a trade-off between throughput and the number of timeouts. Processes that hold locks for an excessive time can reduce the rate of work by causing other applications to wait for locks. This can have a cascade effect similar to a log jam in a river. Reducing the timeout value will cause those application processes that hold locks for excessive times to be exposed by way of the occurrence of timeouts, so that corrective action can be taken. Be aware that the badly behaving transaction is actually not the one likely to be timed out; however, it will allow you to identify the lock holder causing the contention. One example would be a batch process with no intermediate commit points, so that the commit frequency is data volume dependent. On the other hand, causing excessive timeouts requires work to be backed out by DB2 and the application has to retry or restart the process.

Ideally, this value should be set as low as possible without causing an excessive number of timeouts under normal operating conditions. However, the actual value is entirely dependent on the profile of work and so no general guideline can be given.

One practical approach is to find the optimal value for your environment by a process of trial and error, such as:

1. Start with the default of 60 seconds.
2. Monitor the number of timeouts.
3. If none occur and the lock wait time is greater than zero, then reduce the value, say by 5 seconds. Cycle back to 2.
4. If timeouts occur, then identify the cause and correct the process if possible. If the number of timeouts is excessive, then increase the timeout value until the badly behaving transactions have been fixed. Cycle back to 2.



### 7.3.2 DEADLOK (part 1): DEADLOCK TIME

This is the entry DEADLOCK TIME on the DB2 installation panel DSNTIPJ, which corresponds to the IRLM PROC parameter DEADLOK. A deadlock is a situation where two or more requesters are waiting for resources held by the other.

The dead lock time specifies the interval in seconds or milliseconds between each deadlock detection cycle and must be less than the value specified for the IRLM resource wait timeout option. Values between 1 and 5 are interpreted as seconds and between 100 and 5000 as milliseconds. The default value is 1 second.

The dead lock time is the first part of the DEADLOK parameter.

As with the timeout, setting the deadlock detection cycle time is a tradeoff. The deadlock detection process itself can cause latch suspensions; however, if the application design is causing deadlocks, then these need to be resolved in as short a time as possible. Start with the default, and if deadlocks are occurring, then consider reducing the time.

Refer to Example 7-3 on page 205 about using an IRLM modify command to display the current value for the deadlock time. Example 7-5 shows how to dynamically modify the DEADLOK value using an IRLM modify command.

*Example 7-5 How to modify the deadlock time at the IRLM level*

---

```
/MODIFY DB9AIRLM,SET,DEADLOCK=2000
```

```
DXR177I ID9A001 THE VALUE FOR DEADLOCK IS SET TO      2000  MILLISECONDS
```

---

### 7.3.3 DEADLOK (part 2): DEADLOCK CYCLE

This is the entry DEADLOCK CYCLE on the DB2 installation panel DSNTIPJ, which corresponds to second part of the IRLM start procedure DEADLOK parameter. This option is used only for DB2 data sharing and is the number of local deadlock cycles that must expire before the IRLM starts the next global detection cycle. This value is always set to 1 by DB2, irrespective of the value entered on the panel or in the IRLM start procedure.

The DEADLOCK parameter is specified as DEADLOCK TIME,DEADLOCK CYCLE for example 1 (1 is the default). The DEADLOCK CYCLE should be specified as 1 even when not using data sharing.

### 7.3.4 BMPTOUT: IMS BMP TIMEOUT multiplier

This is the entry IMS BMP TIMEOUT on the DB2 installation panel DSNTIPI, which corresponds to the DSNZPARM BMPTOUT. It specifies the number of resource timeout values that an IMS BMP connection is to wait for a lock to be released. The acceptable values are 1 to 254.

The default value is 4. For example, if you use the default value, an IMS BMP connection can wait 4 times the resource timeout value for a resource. This option gives you flexibility in tuning your system to avoid timeouts.

### 7.3.5 DLITOUT: DL/I BATCH TIMEOUT multiplier

This is the entry DL/I BATCH TIMEOUT on the DB2 installation panel DSNTIPI, which corresponds to the DSNZPARM DLITOUT. It specifies the number of resource timeout values that a DL/I batch connection is to wait for a lock to be released. The acceptable values are 1 to 254.

The default is 6. For example, if you use the default value, a DL/I batch application can wait six times the resource timeout value for a resource. This option gives you flexibility in tuning your system to avoid timeouts.

### 7.3.6 UTIMOUT: UTILITY TIMEOUT multiplier

This is the entry UTILITY TIMEOUT on the DB2 installation panel DSNTIP6, which corresponds to the DSNZPARM UTIMOUT. UTILITY TIMEOUT specifies the number of IRLM Resource Wait Timeout values (IRLMRWT) that a utility or DB2 command waits for a lock or for all claims on a resource of a particular claim class to be released. The acceptable values are 1 to 254.

The default is 6.

### 7.3.7 RETLWAIT: RETAINED LOCK TIMEOUT multiplier

This is the entry RETAINED LOCK TIMEOUT on the DB2 installation panel DSNTIPI, which corresponds to the DSNZPARM RETLWAIT. It indicates how long a transaction should wait for a lock on a resource if another DB2 in a data sharing group has failed and is holding an incompatible lock on that resource. The acceptable values are 0 to 254.

The default is 0.

This value is important only in a data sharing environment. Locks that are held by failed DB2 members are called retained locks. The value that you use is a multiplier that is applied to the connection's normal timeout value. For example, if the retained lock multiplier is 2, the timeout period for a call attachment connection that is waiting for a retained lock is  $1 * 2$  (1 for the normal CAF timeout period, and 2 for the additional time that is specified for retained locks).

If you use the default, 0, applications do not wait for incompatible retained locks, but instead the lock request is immediately rejected, and the application receives a resource unavailable SQLCODE. If you have implemented a restart manager, such as z/OS Automatic Restart Manager, and are using the Restart Light for cross system restarts, then consider setting RETLWAIT greater than 0 to wait for retained locks.

### 7.3.8 NUMLKTS: maximum LOCKS PER TABLE (SPACE)

This is the entry LOCKS PER TABLE (SPACE) on the DB2 installation panel DSNTIPJ, which corresponds to the DSNZPARM NUMLKTS. The acceptable values are 0 to 104857600.

The default is 1,000.

You can enter the number of locks as an integer or you can enter a value with a suffix of K or M. If you enter a value with a suffix of K, the number of locks is that value multiplied by 1024. If you enter a value with a suffix of M, the number of locks is that value multiplied by  $1024 \times 1024 = 1,048,576$ .

The value that you specify for this field must be less than the value specified for LOCKS PER USER (NUMLKUS), except when LOCKS PER USER is set to 0.

This value becomes the default value (SYSTEM) for the LOCKMAX clause of the SQL statements CREATE TABLESPACE and ALTER TABLESPACE. A value of 0 indicates that there is no limit to the number of page and row locks that a program can acquire.

Be careful when setting this value to 0, as the minimum storage for IRLM is 2 GB, which corresponds to approximately 3.7 million locks. If an application process were to erroneously acquire that many locks, it could have undesired effects on the ability for IRLM to efficiently process other lock requests.

Refer to 2.2.3, “Lock escalation” on page 30 to learn more about the consequences of choices for this parameter.

There are no guidelines for setting this value, as it is entirely workload dependent. It can be treated in the same way as the timeout parameter: Set a high value initially, and then reduce the value in stages to identify those application processes that take and hold excessive numbers of locks. Unlike the timeout parameter, it is more likely that the badly behaving process will be the one to suffer, as the procedure for lock escalation may well lead to a timeout and backout of all the work, which could be considerable. If the lock escalation is successful, then it is other application processes that will suffer due to the likelihood of lock contention.

### 7.3.9 NUMLKUS: maximum LOCKS PER USER

This is the entry LOCKS PER USER on the DB2 installation panel DSNTIPJ, which corresponds to the DSNZPARM NUMLKUS. It specifies the maximum number of page or row locks that a single application process can hold concurrently on all table spaces. Once that limit is reached, the program that accumulated these locks will terminate with SQLCODE -904.

The maximum includes locks on data pages or rows that the process acquires when it accesses table spaces. The limit applies to all table spaces defined with the LOCKSIZE PAGE, LOCKSIZE ROW, or LOCKSIZE ANY options. The acceptable values are 0 to 104857600 with a default value of 10,000.

A value of 0 means that there is no limit to the number of page and row locks a program can acquire.

You can enter the number of locks as an integer or you can enter a value with a suffix of K or M. If you enter a value with a suffix of K, the number of locks is that value multiplied by 1024. If you enter a value with a suffix of M, the number of locks is that value multiplied by  $1024 \times 1024 = 1,048,576$ .

DB2 assumes that each lock requires 540 bytes of storage. If you define referential constraints between columns, you might want to select a higher value for this field.

To avoid exhausting the IRLM's storage for locks, follow these guidelines:

- ▶ Do not specify 0 or a very large value unless it is specifically required to run an application.
- ▶ Consider the design of your applications. Long-running applications, particularly those that perform row-level locking, have few or infrequent commit points or use repeatable-read isolation, and may use substantial amounts of lock storage. You should perform frequent commits to release locks.

These values are constraints for a single application. Each concurrent application can hold the maximum number of locks specified here.

There are no guidelines for setting this value, as it is entirely workload dependent. It can be treated in the same way as the timeout parameter: Set a high value initially and then reduce the value in stages to identify those application processes that take and hold excessive numbers of locks. Unlike the timeout parameter, the badly behaving process *will* be the one to suffer, because once the limit is reached, the application process is terminated, all work done so far will be backed out, and all locks released.

### 7.3.10 RRULOCK: U LOCK FOR RR/RS isolation

This is the entry U LOCK FOR RR/RS on the DB2 installation panel DSNTIPI, which corresponds to the DSNZPARM RRULOCK. It specifies whether DB2 will use a U lock when the isolation of the program is repeatable read (RR) or read stability (RS). This is applicable for these SQL statements:

- ▶ SELECT with FOR UPDATE OF.
- ▶ UPDATE and DELETE, without a cursor.
- ▶ If the value specified is YES, the lock used in these cases is update (U). If no update occurs, the U lock is demoted to an S lock on the next fetch. If an update does occur, the U lock is promoted to an X lock with COMMIT duration. The acceptable values are YES or NO.
- ▶ If the value specified is NO (the default), the lock used in this case is a shared (S) lock.

If your applications make frequent updates with RR or RS isolation, the U lock reduces the potential for deadlocks. In read-only processes or when updates are less frequent, S locks generally provide more concurrency and better performance (reduced CPU time). If you want concurrency with a mix of read and write, use the value NO. If some application processes need the more restrictive lock due to frequency of deadlocks being experienced, then specify the USE AND KEEP EXCLUSIVE/UPDATE/SHARE LOCKS clause on those statements requiring the taking and keeping of locks until COMMIT.

### 7.3.11 XLKUPDLT: X LOCK FOR SEARCHED Update or Delete

This is the entry X LOCK FOR SEARCHED Update or Delete on the DB2 installation panel DSNTIPI, which corresponds to the DSNZPARM XLKUPDLT. It specifies the locking method that is to be used when performing a searched update or delete. The acceptable values are YES, NO, or TARGET, with NO as the default.

- ▶ A value of NO means DB2 uses an S or U lock when scanning for qualifying rows. For any qualifying rows or pages, the lock is upgraded to an X lock before performing the update or delete. For non-qualifying rows or pages, the lock is released if ISOLATION(CS) is used. For ISOLATION(RS) or ISOLATION(RR), an S lock is retained on the rows or pages until the next commit point. Use this option to achieve higher rates of concurrency.
- ▶ A value of YES means DB2 uses an X lock on qualifying rows or pages based on stage 1 predicates. For ISOLATION(CS), the lock is released if the rows or pages are not updated or deleted. For ISOLATION(RS) or ISOLATION(RR), an X lock is retained until the next commit point. A value of YES is beneficial in a data sharing environment when most or all searched updates and deletes use an index. If YES is specified and searched, updates or deletes result in a table space scan, and the likelihood of timeouts and deadlocks greatly increases.

- A value of TARGET means DB2 combines YES and NO behavior. DB2 uses an X lock on qualifying rows or pages of the specific table that is targeted by the update or delete statement. DB2 uses an S or U lock when scanning for rows or pages of other tables that are referenced by the query (for example, tables that are referenced only in the WHERE clause of the query). For non-qualifying rows or pages, the lock is released if ISOLATION(CS) is used. For ISOLATION(RS) or ISOLATION(RR), an S lock is retained on the rows or pages until the next commit point.

### 7.3.12 EVALUNC: EVALUATE UNCOMMITTED rows

This is the entry EVALUATE UNCOMMITTED on the DB2 installation panel DSNTIP8, which corresponds to the DSNZPARM EVALUNC. It specifies whether predicate evaluation can occur on uncommitted data of other application processes. Acceptable values are YES and NO, with NO as the default.

The option applies only to stage 1 predicate processing that uses table access (table space scan, index-to-data access, and RID-list processing) for queries with isolation level RS or CS.

Although the option influences whether predicate evaluation can occur on uncommitted data, it does not influence whether uncommitted data is returned to an application. Queries with isolation level RS or CS return only committed data. They never return the uncommitted data of other transactions, even if predicate evaluation occurs. If data satisfies the predicate during evaluation, the data is locked as needed, and the predicate is re-evaluated as needed before the data is returned to the application.

- If you specify NO, the default predicate evaluation occurs only on committed data (or on the application's own uncommitted changes). NO ensures that all qualifying data is always included in the answer set.
- If you specify YES, predicate evaluation can occur on uncommitted data of other transactions. With YES, data might be excluded from the answer set. Data that does not satisfy the predicate during evaluation, but then, because of undo processing (ROLLBACK or statement failure), reverts to a state that does satisfy the predicate, is missing from the answer set. A value of YES enables DB2 to take fewer locks during query processing. The number of avoided locks depends on:
  - The query's access path
  - The number of evaluated rows that do not satisfy the predicate
  - The number of those rows that are on overflow pages

Specify YES to improve concurrency if your applications can tolerate returned data that might falsely exclude any data that would be included as the result of undo processing (ROLLBACK or statement failure).

### 7.3.13 SKIPUNCI: SKIP UNCOMMITTED INSERTS

This is the entry SKIP UNCOMM INSERTS on the DB2 installation panel DSNTIP8, which corresponds to the DSNZPARM SKIPUNCI. This option specifies whether statements ignore a row that was inserted by a transaction (other than itself) and that has not yet been committed or aborted. The acceptable values are NO (which is the default) or YES.

This parameter applies only to statements running with row-level locking and isolation level read stability (RS) or cursor stability (CS):

- ▶ If you specify the default behavior of NO, DB2 waits for the inserted row to be committed or rolled back, then processes the row as a qualifying row if the insert commits or moves on to find another row if the insert is rolled back. If a transaction performs one or more inserts, then spawns a second transaction, specify NO for SKIP UNCOMM INSERTS if the first transaction needs the second transaction to wait for the outcome of the inserts.
- ▶ If you specify a value of YES, DB2 behaves as though the uncommitted row has not yet arrived and the row is skipped. Specifying a value of YES offers greater concurrency than the default value of NO.

### 7.3.14 IDTHTOIN: IDLE THREAD TIMEOUT field

This is the entry IDLE THREAD TIMEOUT field on installation panel DSNTIPR, which corresponds to the DSNZPARM IDTHTOIN. It specifies the approximate time, in seconds, that an active server thread should be allowed to remain idle before it is canceled. The acceptable values are 0 to 9999. The default is 120.

The thread is canceled after the timeout value expires; its locks and cursors are released. Inactive and in-doubt threads are not subject to timeout. The value that you specify for DDF THREADS determines whether a thread can become inactive, and thus are not subject to a timeout.

Threads are checked every two minutes to see if they have exceeded the timeout value. If the timeout value is less than two minutes, the thread might not be canceled if it has been inactive for more than the timeout value but less than two minutes.

Specifying 0 disables timeout processing. If timeout processing is disabled, idle server threads remain in the system and continue to hold their resources, if any.

### 7.3.15 LRDRTHLD: LONG-RUNNING READER field

This is the entry LONG-RUNNING READER field on installation panel DSNTIPE, which corresponds to the DSNZPARM LRDRTHLD. It specifies the number of minutes that a read claim can be held by an agent before DB2 writes a trace record to report it as a long-running reader.

The acceptable values are 0 to 1439 minutes. The default is 0.

If you specify a value of 0, DB2 will not report long-running readers. Refer to “Holding a read claim for more than LRDRTHLD minutes without a commit” on page 250 for a detailed explanation.

### 7.3.16 URCHKTH: UR CHECK FREQ field

This is the entry UR CHECK FREQ field on installation panel DSNTIPL, which corresponds to the DSNZPARM URCHKTH. It specifies the number of checkpoint cycles that are to complete before DB2 issues a warning message to the console and instrumentation for an uncommitted unit of recovery (UR). The acceptable values are 0 to 255.

The default is 0.

If you specify a value of 0, DB2 will not report long-running units of recovery. Refer to “DSNR035I: uncommitted UR for a number of system checkpoints” on page 226 for a detailed explanation.

### **7.3.17 URLGWTH: UR LOG WRITE CHECK field**

This is the entry UR LOG WRITE CHECK field on installation panel DSNTIPL, which corresponds to the DSNZPARM URLGWTH. It specifies the number of log records that are to be written by an uncommitted unit of recovery (UR) before DB2 issues a warning message to the console. The acceptable values are 0 to 1000K.

The default is 0.

If you specify a value of 0, DB2 will not report long-running units of recovery. The purpose of this option is to provide notification of a long-running UR. Long-running URs might result in a lengthy DB2 restart or a lengthy recovery situation for critical tables. Specify the value in 1-K (1000 log records) increments. A value of 0 indicates that no write check is to be performed. Refer to “Exceeding URLGWTH number of log records without a commit” on page 248 for a detailed explanation.

### **7.3.18 PCLOSEN: RO SWITCH CHKPTS field**

This is the entry RO SWITCH CHKPTS field on installation panel DSNTIPL, which corresponds to the DSNZPARM PCLOSEN. This parameter indicates the number of consecutive DB2 checkpoints since a set or partition was last updated, after which DB2 converts the set or partition from read-write to read-only. The acceptable values are 0 to 32767.

The default is 5.

This value is used in conjunction with RO SWITCH TIME. If the condition for RO SWITCH TIME or RO SWITCH CHKPTS is met, the set or partition is converted from read-write to read-only.

For NOT LOGGED table spaces, DB2 converts the set or partition from read-write to read-only after one checkpoint, regardless of the value of RO SWITCH CHKPTS.

Having DB2 switch an infrequently updated set from read-write to read-only can be a performance benefit for recovery, logging, and for data sharing processing.

The settings for this parameter is discussed in 10.8.4, “Pseudo-close considerations” on page 339.

### **7.3.19 PCLOSET: RO SWITCH TIME field**

This is the entry RO SWITCH TIME field on installation panel DSNTIPL, which corresponds to the DSN ZPARM PCLOSET. This parameter Indicates the number of minutes since a set or partition was last updated, after which DB2 converts the set or partition from read-write to read-only. The acceptable values are 0 to 32767.

The default is 10 (minutes).

This value is used in conjunction with RO SWITCH CHKPTS. If the condition for RO SWITCH CHKPTS or RO SWITCH TIME is met, the set or partition is converted from read-write to read-only.

For NOT LOGGED table spaces, DB2 converts the set or partition from read-write to read-only after one minute, regardless of the value of RO SWITCH TIME.

Having DB2 switch an infrequently updated set from read-write to read-only can be a performance benefit for recovery, logging, and for data sharing processing.

The settings for this parameter is discussed in 10.8.4, “Pseudo-close considerations” on page 339.

### 7.3.20 CHKFREQ: CHECKPOINT FREQ field

This is the entry CHECKPOINT FREQ field on installation panel DSNTIPL, which corresponds to the DSNZPARM CHLFREQ. Specify the system checkpoint frequency in minutes or in number of log records. If you have widely variable logging rates, maximize system performance by specifying the checkpoint frequency in time. The acceptable values are 200 to 16000000 (log records) or 1 to 60 (minutes).

The default is 500000 records.

DB2 starts a new checkpoint at the interval you specify, either in minutes, or in the number of log records.

You can use the SET LOG command to dynamically change the number of log records between checkpoints.

If your primary concern is DB2 restart time, use a checkpoint frequency between 200,000 and 1,000,000 log records; otherwise, use a checkpoint frequency of 2 to 5 minutes.

## 7.4 DB2 system parameters that indirectly affect locking

The following DSNZPARMs have indirect consequences for locking and so are included here for completeness.

### 7.4.1 SMFACCT: SMF ACCOUNTING field

This is the entry SMF ACCOUNTING field on installation panel DSNTIPN, which corresponds to the DSNZPARM SMFACCT. It specifies whether DB2 is to send accounting data to SMF automatically when DB2 is started. This field also specifies what classes are sent. The acceptable values are YES, NO, list of classes, or an asterisk (\*). The default is 1.

- ▶ NO  
Specifies no automatic start of classes.
- ▶ YES  
Starts the trace for the default class (class 1). You might also need to update the SMFPRMxx member of SYS1.PARMLIB to permit SMF to write the records.
- ▶ List of classes  
Starts the specified classes. Enter a list of class numbers (any integer from 1 to 32), separated by commas. Only classes 1 to 5, 7, 8, and 10 are defined by DB2.
- ▶ An asterisk (\*)  
Starts all classes.



## 7.4.2 SMFSTAT: SMF STATISTICS field

This is the entry SMF STATISTICS field on installation panel DSNTIPN, which corresponds to the DSNZPARM SMFSTAT. It specifies whether DB2 is to send statistical data to SMF automatically when DB2 is started. This field also specifies what classes are sent. The acceptable values are Yes, No, list of classes, or asterisk(\*):

- ▶ NO  
No automatic start.
- ▶ YES  
The default. It starts the trace for the classes 1, 3, 4, 5, and 6. You might also need to update the SMFPRMxx member of SYS1.PARMLIB to permit SMF to write the records.
- ▶ List of classes  
It starts the specified classes. This is a list of class numbers (any integer from 1 to 32), separated by commas. Classes 1 to 29 are reserved for use by DB2 with classes 1, 2, 3, 4, 5, 6, and 8 being defined. Note that IFCID 225 is now part of class 1, but can still be traced by class 6 if required, and that class 5 is only used for data sharing.  
You might want to add class 8 to the default values statistics to track IFCID 199, that is, data set I/O statistics.
- ▶ An asterisk (\*)  
Starts all classes.

## 7.4.3 STATIME: STATISTICS TIME field

This is the entry STATISTICS TIME field on installation panel DSNTIPN, which corresponds to the DSNZPARM STATIME. It specifies the time interval, in minutes, between statistics collections. Statistics records are written approximately at the end of this interval.

The range of acceptable values is 0 to 1440 with a default of 5.

Our recommendation is to set this to 1 minute. The records can be consolidated for reporting purposes to whatever interval is desired, such as 15 minutes. The records at 1 minute are then still available for analysis, such as during periods when the system is temporarily under stress.

## 7.4.4 SYNCVAL: STATISTICS SYNC field

This is the entry STATISTICS SYNC field on installation panel DSNTIPN, which corresponds to the DSNZPARM SYNCVAL. It specifies whether DB2 statistics recording is to be synchronized with some part of the hour.

- ▶ NO  
The default. It specifies that statistics recording is not synchronized. NO is the default.
- ▶ 0 to 59  
The DB2 statistics recording interval is synchronized with the beginning of the hour (0 minutes past the hour) or with any number of minutes past the hour up to 59. This parameter has no effect if STATIME is greater than 60.  
Use this parameter to synchronize the measurements with the output from RMF interval reports.





## Part 3


# Monitoring and problem determination

In this part, we discuss the types of concurrency and locking problems you may encounter, how to check for them, and how to analyze them.

This part contains the following chapters:

- ▶ Chapter 8, “Identifying locking and concurrency problems” on page 219
- ▶ Chapter 9, “Analyzing concurrency problems” on page 255





## Identifying locking and concurrency problems

In this chapter, we discuss the types of concurrency and locking problems you may encounter and show ways to check for them.

The following topics are covered:

- ▶ Different types of locking and concurrency problems
- ▶ Concurrency warning signs
- ▶ Periodic monitoring for concurrency problems

In order to create lock contention issues, we used the environment and the workloads described in Appendix A, “System topology and workload” on page 441.

## 8.1 Different types of locking and concurrency problems

In this section, we briefly introduce the different kinds of locking and concurrency problems that can occur, and provide some references to where you can discover additional information about them. We discuss how to analyze them in Chapter 9, “Analyzing concurrency problems” on page 255.

Locking and concurrency problems can be categorized as follows:

- ▶ Timeouts
- ▶ Deadlocks
- ▶ Lock escalation
- ▶ Long suspension times
- ▶ Excessive number of locks acquired

### 8.1.1 Timeouts

Timeouts are easy to identify. DB2 writes messages to the console indicating a timeout occurred. Refer to “DSNT376I: timeout message” on page 223 for details.

If the DB2 statistics trace class 3 or performance trace class 6 is active, an IFCID 196 trace record is produced. It contains detailed information about the applications and the resources involved in the timeout. Refer to “IFCID196: timeout information” on page 246 for details.

For a general discussion on what a timeout is and when it occurs, refer to Chapter 2, “Transaction locking” on page 13 and Chapter 9, “Analyzing concurrency problems” on page 255, respectively.

### 8.1.2 Deadlocks

Deadlocks are also easy to identify. DB2 writes messages to the console indicating a deadlock occurred. Refer to “DSNT375I: deadlock message” on page 222 for details.

If the DB2 statistics trace class 3 or performance trace class 6 is active, an IFCID 172 trace record is produced. It contains detailed information about the applications and resources involved in the deadlock. Refer to “IFCID 172: deadlock information” on page 243 for its contents.

For a general discussion about what a deadlock is and when it occurs, refer to Chapter 2, “Transaction locking” on page 13 and Chapter 9, “Analyzing concurrency problems” on page 255, respectively.

### 8.1.3 Lock escalation

Lock escalation can have a large impact on the system’s throughput. Lock escalation is a tradeoff between performance (by having to acquire and manage fewer locks) and concurrency (mainly reducing the ability for other users in the system to access the object whose locks got escalated.)

Lock escalation is also easy to track, as it produces console messages that you can act upon. These are discussed in “DSNI031I: lock escalation” on page 225. If the DB2 statistics trace class 3 or performance trace class 6 is active, an IFCID 337 trace record is produced. It contains detailed information about the application and resource involved in the lock escalation. Refer to “IFCID 337: lock escalation” on page 250 for details about how to interpret this information.

## 8.1.4 Long suspension times

When the throughput of an application is below expectations, it is always good to check whether this is caused by long lock/latch suspensions.

Using the DB2 accounting information, either at the plan or at the package level, you can check to see whether an application suffers from excessive lock/latch suspensions.

Example 8-1 shows an excerpt of an OMEGAMON PE accounting report.

*Example 8-1 Plan level CLASS 3 suspensions in a DB2 accounting report*

CLASS 3 SUSPENSIONS	ELAPSED TIME	EVENTS
-----	-----	-----
<b>LOCK/LATCH (DB2+IRLM)</b>	5.615281	293452
SYNCHRON. I/O	22.570365	5245
DATABASE I/O	1.721747	778
LOG WRITE I/O	20.848617	4467
OTHER READ I/O	0.430458	33
OTHER WRTE I/O	0.200865	59
SER.TASK SWTCH	5.854532	172
UPDATE COMMIT	0.367529	66
OPEN/CLOSE	2.008859	31
SYSLGRNG REC	0.193351	32
EXT/DEL/DEF	3.194758	12
OTHER SERVICE	0.090036	31
ARC.LOG(QUIES)	0.000000	0
LOG READ	0.000000	0
<b>DRAIN LOCK</b>	0.060113	116
<b>CLAIM RELEASE</b>	0.000000	0
<b>PAGE LATCH</b>	0.036056	29
<b>NOTIFY MSGS</b>	0.000000	0
<b>GLOBAL CONTENTION</b>	11.456154	11962
COMMIT PH1 WRITE I/O	0.000000	0
ASYNCH CF REQUESTS	12.538104	16451
TCP/IP LOB	0.000000	0
TOTAL CLASS 3	58.761929	327519

The highlighted sections are fields that you may need to look at when investigating concurrency problems. They are discussed in “Accounting class 1,2,3 times” on page 237.

## 8.1.5 Excessive number of locks acquired

Sometimes an application does not really suffer from any concurrency issues, especially when there is little other concurrent activity against the objects, but it is still worth to have a look at the number of locks that are acquired by an application. Acquiring a lock requires CPU cycles, so any lock request that can be avoided can help reduce the CPU usage of the application.

In addition, when the workload increases over time and more concurrent access to these objects is required, if the application acquires a large number of locks, the chances that you will run into a concurrency problem at that time are much higher.

## 8.2 Concurrency warning signs

As concurrency problems can have a dramatic impact on the throughput of a system, we recommend watching out for any warning signs that indicate a concurrency problem or that things are building up towards a concurrency issue.

The warnings can be built by the application or sent by DB2 to the console and intercepted by automation.

### 8.2.1 User warnings

It is always wise to build some intelligence into a program that allows you to alert users or administrators in order to track how well the program is running. This is particularly interesting when the application is coded to use an optimistic locking approach. It is the program that checks to see whether a row has changed between reading it and updating it. As such, the program may have to retry several times before it actually manages to update the row. In such a scenario, these attempts do not result in many or long lock suspensions and may be difficult to spot in a DB2 accounting report. It is therefore a good coding practice to track the number of retries in the application and write a nice message at the end of the run. This is more practical to do in batch types applications than for online transactions.

### 8.2.2 Console messages

DB2 is very much aware of the effects of concurrency problems and tries to help by alerting you by way of a number of console messages when the system is experiencing a problem. An automation process can then use these messages to analyze the problem and alert the right people. The following set of messages related to concurrency problems or potential problems are issued by DB2. Refer to 8.3, “Periodic monitoring for concurrency problems” on page 227 for information about how to set up monitoring for this type of message or its associated trace records.

#### **DSNT375I: deadlock message**

When a deadlock occurs, a message like the one shown in Example 8-2 is written to the console.

*Example 8-2 DSNT375I message*

---

```
DSNT375I  -DB9A PLAN=DSNREXX WITH 646
CORRELATION-ID=BARTR4
CONNECTION-ID=DB2CALL
LUW-ID=USIBMSC.SCPDB9A.C483AD1BC332=212
THREAD-INFO=BART:*:*:*
IS DEADLOCKED WITH PLAN=DSNREXX WITH
CORRELATION-ID=BARTR3
CONNECTION-ID=DB2CALL
LUW-ID=USIBMSC.SCPDB9A.C483AD1530A1=211
THREAD-INFO=BART:*:*:*
ON MEMBER DB9A
```

---



The message contains the plan name, correlation-ID, connection-ID, and LUW-ID of the program that was the victim of the deadlock (BARTR4 job in Example 8-2 on page 222). It also indicates one of the other threads that are involved in the deadlock (BARTR3 job in Example 8-2 on page 222). There can be more than one other thread involved in the deadlock. In order to get the complete picture of the deadlock, you need to analyze the IFCID 172 trace record.

Following the DSNT375I message, there is also a DSNT501I message that has information about the resources involved in the deadlock.

### **DSNT376I: timeout message**

When a timeout occurs, a message like the one shown in Example 8-3 is written to the console.

#### *Example 8-3 DSNT376I message*

---

```
DSNT376I  -DB9A PLAN=DSNREXX WITH 932
            CORRELATION-ID=BARTR1
            CONNECTION-ID=DB2CALL
            LUW-ID=USIBMSC.SCPDB9A.C4825FC3FB09=1896
            THREAD-INFO=BART:*:*:*
IS TIMED OUT. ONE HOLDER OF THE RESOURCE IS PLAN=DSNREXX WITH
            CORRELATION-ID=BARTR2
            CONNECTION-ID=DB2CALL
            LUW-ID=USIBMSC.SCPDB9A.C4826081A772=1899
            THREAD-INFO=BART:*:*:*
            ON MEMBER DB9A
```

---

The message has the plan name, correlation-ID, connection-ID, and LUW-ID of the program that timed out (BARTR1 job in Example 8-3). It also indicates one of the threads that is holding the lock on which you timed out (BARTR2 job in Example 8-3). There could be more than one plan holding this resource. To have the complete picture, you need to analyze the IFCID 196 trace record.

Following the DSNT376I message, there is also a DSNT501I message that has information about the resource that is involved in the timeout.

## DSNT501I: resource unavailable message

The DSNT501I message shown in Example 8-4 provides information about a resource that is unavailable to a transaction. A resource can be unavailable for many reasons. The actual reason why the resource is not available is provided after the REASON keyword. A detailed explanation of all DB2 reason codes can be found in *DB2 Version 9.1 for z/OS Codes*, GC18-9843 manual.

Example 8-4 DSNT501 message

---

```
DSNT501I  -DB9A DSNILMCL RESOURCE UNAVAILABLE 647
          CORRELATION-ID=BARTR4
          CONNECTION-ID=DB2CALL
          LUW-ID=USIBMSC.SCPDB9A.C483AD1BC332=212
          REASON 00C90088
          TYPE 00000302
          NAME GLWBART .GLWSDPT .X'000002'

DSNT501I  -DB9A DSNILMCL RESOURCE UNAVAILABLE 933
          CORRELATION-ID=BARTR1
          CONNECTION-ID=DB2CALL
          LUW-ID=USIBMSC.SCPDB9A.C4825FC3FB09=1896
          REASON 00C9008E
          TYPE 00000D01
          NAME 00000752.00000003
```

---

In Example 8-4, reason code (RC) *00C90088* is related to a deadlock and *00C9008E* is related to a timeout. The details for the reason codes are shown in Example 8-5.

Example 8-5 00C900E8 and 00C9008E reason codes

---

### 00C90088

Explanation: The resource identified by NAME in message DSNT501I is involved in a deadlock condition.

### 00C9008E

Explanation: A lock request for the resource identified by NAME could not be granted, and the request waited for a period longer than the maximum specified by the installation.

---

The different resource TYPEs of the DSNT501I message can be found in Appendix F, “Resource types”, in *DB2 Version 9.1 for z/OS Codes*, GC18-9843 and in Appendix C, “Resource types” in *DB2 Version 9.1 for z/OS Messages*, GC18-9849.

- ▶ TYPE 302 is a *table space* page in the format DB.SP.PG, so in Example 8-4, it is page x'2' of the GLWBART.GLWSDPT *table space*.
- ▶ TYPE D01 indicates that the lock is on an DBID/OBID and that the resource name will be an DBID/OBID pair (in *decimal*). You can then use these values do a lookup in the DB2 catalog to discover the resource that is not available to your transaction. In this particular example, the timeout occurred on a table, so the OBID is a table OBID and you need to do the lookup against the *SYSIBM.SYSTABLES* catalog table.

There are many other types of resource unavailable reason codes that are not related to concurrency, but we do not describe them in this publication.

## DSNI031I: lock escalation

Example 8-6 shows the console message issued by DB2 when lock escalation occurs.

*Example 8-6 DSNIO31I message*

---

**DSNIO31I** -DB9A DSNILKES - **LOCK ESCALATION HAS** 843

**OCCURRED FOR**

RESOURCE NAME = GLWBART.GLWSEMP  
LOCK STATE = S  
PLAN NAME : PACKAGE NAME = DSNEPC : DSNEPC68  
COLLECTION-ID = DSNEPC  
STATEMENT NUMBER = 000251  
CORRELATION-ID = BART  
CONNECTION-ID = TSO  
LUW-ID = USIBMSC.SCPDB9A.C483CF95A128  
THREAD-INFO = BART : \* : \* : \*

---

The message contains the object on which the escalation occurred (GLWBART.GSWSEMP) and the lock state to which it was escalated. In Example 8-6, this was escalated from IS to S. The message also provides the necessary information to identify the thread and the SQL statement number that issued the statement; in this case, it was the SPUFI program, so the statement number is not very useful.

Similar information can be found in the IFCID 337 trace record. This record is written when statistics trace class 3 is active or when performance trace class 6 is active.

## DSNJ031I: uncommitted UR with many log records

Example 8-7 shows the DSNJ031I message sent to the console when a unit of recovery has written more log records than indicated by the URLGWTH DSNZPARM value without committing.

In this DB2 subsystem (DB9A), URLGWTH is set to 100, meaning 100,000 log records. The default value for URLGWTH is zero, which means that DB2 will not monitor for URs that do a great deal of logging without intermediate commits.

*Example 8-7 DSNJ031I message*

---

**DSNJ031I** -DB9A DSNJW001 WARNING - **UNCOMMITTED UR** 545

**HAS WRITTEN 100000 LOG RECORDS -**

CORRELATION NAME = BART  
CONNECTION ID = TSO  
LUWID = USIBMSC.SCPDB9A.C484A9A68271 = 738  
PLAN NAME = DSNEPC  
AUTHID = BART  
END USER ID = \*  
TRANSACTION NAME = \*  
WORKSTATION NAME = \*

---

The message also contains information that allows you to identify the thread (like correlation name, authorization ID, and plan name) that is doing a great deal of logging without committing.

As there is normally a strong relationship between the number of log records that are written and the number of locks that are acquired, there is a good chance that this “problem” transaction may also cause concurrency problems and needs to be investigated.

Similar information can be found in the IFCID 313 trace record. This record is written when statistics trace class 3 is active.

### **DSNR035I: uncommitted UR for a number of system checkpoints**

A DSNR035I console message, shown in Example 8-8, indicates that there is an in-flight unit of recovery that has not committed for more than URCHKTH system checkpoints. URCHKTH DSNZPARM controls the number of system checkpoints a UR has to be in flight before the DSNR035I message is produced. URCHKTH is set to 3 on our system. Under the assumption that you normally take a system checkpoint every 2 to 5 minutes, a warning message after 6 to 15 minutes is reasonable. But, as with all DSNZARM settings, you should set the value based on your specific environment.

*Example 8-8 DSNR035I message*

---

```
DSNR035I  -DB9A DSNRPBCW WARNING - UNCOMMITTED UR 662
AFTER 3 CHECKPOINTS -
CORRELATION NAME = BART
CONNECTION ID   = TSO
LUWID = USIBMSC.SCPDB9A.C484B098D990 = 909
PLAN NAME = DSNESPCS
AUTHID = BART
END USER ID = *
TRANSACTION NAME = *
WORKSTATION NAME = *
```

---

If there is long, in-flight unit of recovery, there will also be some X locks held for an extended period of time. This increases the chance that somebody else needs to access any of these resources that are currently being held by this long running UR.

As with the other console messages, DB2 provides the necessary information that allows you to identify the long running unit of work and to analyze why the program is not issuing a commit for such an extended period of time.

Similar information about long running URs can be found in the IFCID 313 trace record. This record is written when statistics trace class 3 is active.

There is a similar message (DSNR036I) for outstanding in-doubt URs at system checkpoint time. Example 8-9 shows a sample message.

*Example 8-9 DSNR036I message*

---

```
DSNR036I csect-name UNRESOLVED UR
ENCOUNTERED DURING CHECKPOINT -
CORRELATION NAME = xxxxxxxxxxxx
CONNECTION ID = yyyyyyyy
LUWID = logical-unit-of-work-id
PLAN NAME = xxxxxxxx
AUTHID = xxxxxxxx
```

---

As in-doubt URs can be holding locks, they may also affect concurrency, and they need to be resolved as soon as possible.

## 8.3 Periodic monitoring for concurrency problems

Even after an application has moved to production, it is important to keep monitoring the application. Over time, the behavior may change, for example, because the workload increases or the data gets disorganized. Because we are discussing DB2 concurrency and serialization, in this section we focus on monitoring for locking and concurrency problems.

We discuss the following topics:

- ▶ Which information to gather
- ▶ Analyzing DB2 statistics data
- ▶ Analyzing DB2 accounting data
- ▶ Periodic monitoring of DB2 system level concurrency related IFCIDs
- ▶ Exception monitoring using OMEGAMON PE

### 8.3.1 Which information to gather

Most users run with a set of standard active DB2 traces. The information is often used for chargeback purposes, but it contains a wealth of information that can also be used to check the health of the system.

We recommend having the following traces permanently active on all your DB2 systems:

- ▶ DB2 statistics trace classes
  - 1: System wide information about the work performed by the DB2 system.
  - 3: Information about deadlocks, timeouts, lock escalations, and long running units of work. This trace class is extremely valuable in identifying concurrency problems, as it activates IFCID 172 (deadlock), 196 (timeout), 337 (lock escalation), and 313 (long running URs).
  - 4: DDF exception conditions.
  - 5:- Data sharing statistics.

**Tip:** STATIME DSNZPARM determines the interval at which DB2 writes out its statistics information for classes 1 and 5. The default value is 5 minutes. However, we now recommend using STATIME=1. The cost of gathering this information is negligible and it provides extremely valuable information to analyze any kind of slowdown performance problems.

- ▶ DB2 accounting trace classes
  - 1: Total Elapsed Time (ET) and CPU time of the thread/plan and many useful counters.
  - 2: In addition to the total time, the time (ET and CPU) spent inside DB2 is collected.

This trace class is more expensive (typically around 2.5% for online transactions) than gathering class 1 information. It can create some overhead, especially for applications that issue many DB2 requests, like fetch intensive applications (up to 10% in heavy batch environments).

However, this information is required to determine whether the time is spent in DB2 or elsewhere, and is therefore extremely valuable.

So, unless you really have a CPU issue, activate accounting class 2 all the time. If that is not possible, activate it for one hour each day during a high activity period.

- 3: Activating this trace class adds an additional level of granularity, as it allows you to determine when DB2 has to wait for something, for example, for a lock to become available, how many times that occurs, and how long DB2 waited for it. You need this information to analyze locking problems.

Gathering accounting class 3 information is inexpensive, so it is normally not a problem to keep it active all the time. In cases where transactions experience a very high number of lock/latch contentions, tracking them can cause a noticeable impact. If that is the case, you can disable accounting trace class 3.

- 7: Similar to accounting class 2, information about the ET and CPU time used is collected, but at the package/DBRM level.
- 8: Similar to accounting class 3 information, but at the package/DBRM level.

Both traces can be started at DB2 startup time by way of SMFSTAT (statistics) and SMFACCT for accounting DSNZPARMs.

The DB2 statistics and accounting information allow you to do a periodic check of your applications to determine whether they are still performing to the service level agreements, including a health check of their locking and concurrency behavior.

### 8.3.2 Analyzing DB2 statistics data

Let us start with analyzing the DB2 statistics data. This information is subsystem wide or member wide if you are running in a data sharing environment. We discuss the sections that have information that is pertinent to locking, latching, concurrency, or serialization.

We only discuss the counters related to non-data sharing. For information about locking and serialization specific to the DB2 data sharing environment, refer to Chapter 11, “Monitoring data sharing locking activity” on page 365.

We use the IBM Tivoli® Omegamon XE Performance Expert for DB2 for z/OS V4.2 (OMEGAMON PE) statistics batch reports. Other monitors can have a different layout, but the data should be pretty similar, as the information is normally based on the statistics records produced by DB2.

The OMEGAMON PE statistics report shows the total number of events of a certain type (QUANTITY) that occurred in the interval that is being reported. Without knowing the elapsed time of the interval in which these events occurred, the total is not very meaningful. For this reason, OMEGAMON PE also provides information per second (/SECOND), per thread (/THREAD) and per commit (/COMMIT).

When you are monitoring a DDF workload, as is the case in Example 8-10 (this is an excerpt from a WebSphere Application Server Trade6<sup>1</sup> run by way of a T4 connection into DDF), the /THREAD information is not very useful, as OMEGAMON PE does not consider DBATs as threads. Only “local” threads are counted (see CREATE THREAD counter in the statistics report).

*Example 8-10 Statistics highlights: subsystem services in remote location sections*

---

---- HIGHLIGHTS -----			
INTERVAL START :	07/23/09 19:05:15.11	SAMPLING START:	07/23/09 19:05:15.11
INTERVAL END :	07/23/09 19:16:43.29	SAMPLING END :	07/23/09 19:16:43.29
INTERVAL ELAPSED:	11:28.180214	OUTAGE ELAPSED:	0.000000
		<b>TOTAL THREADS</b> :	5.00
		<b>TOTAL COMMITS</b> :	14846.00
		DATA SHARING MEMBER:	N/A

SUBSYSTEM SERVICES	QUANTITY
-----	-----
IDENTIFY	8.00
<b>CREATE THREAD</b>	<b>5.00</b>
SIGNON	0.00
TERMINATE	15.00
<b>ROLLBACK</b>	<b>1.00</b>
COMMIT PHASE 1	0.00
COMMIT PHASE 2	0.00
READ ONLY COMMIT	0.00
UNITS OF RECOVERY INDOUBT	0.00
UNITS OF REC.INDBT RESOLVED	0.00
<b>SYNCHS(SINGLE PHASE COMMIT)</b>	<b>10.00</b>
QUEUED AT CREATE THREAD	0.00
SUBSYSTEM ALLIED MEMORY EOT	0.00
SUBSYSTEM ALLIED MEMORY EOM	0.00

DRDA REMOTE LOCS	SENT	RECEIVED
-----	-----	-----
TRANSACTIONS	0.00	48.00
CONVERSATIONS	0.00	48.00
CONVERSATIONS QUEUED	0.00	
SQL STATEMENTS	0.00	74065.00
<b>SINGLE PHASE COMMITS</b>	<b>0.00</b>	<b>14835.00</b>
SINGLE PHASE ROLLBACKS	0.00	0.00

---

When calculating the number of commits (as a denominator for the /COMMIT counters), commit requests from remote locations are counted. In other words, commits from DBATs are included. So for this workload, /COMMIT is a more meaningful measure than /THREAD.

<sup>1</sup> You can download the Trade6 workload and its description from the following URL:  
<https://www.software.ibm.com/webapp/iwm/web/preLogin.do?source=trade6>

## Statistics locking activity

Example 8-11 shows the non-data sharing locking activity section of the DB2 statistics report.

*Example 8-11 Non-data sharing statistics locking activity*

LOCKING ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
SUSPENSIONS (ALL)	752.00	1.09	150.40	0.05
SUSPENSIONS (LOCK ONLY)	364.00	0.53	72.80	0.02
SUSPENSIONS (IRLM LATCH)	379.00	0.55	75.80	0.03
SUSPENSIONS (OTHER)	9.00	0.01	1.80	0.00
TIMEOUTS	0.00	0.00	0.00	0.00
DEADLOCKS	0.00	0.00	0.00	0.00
LOCK REQUESTS	521.9K	758.37	104.4K	35.15
UNLOCK REQUESTS	26515.00	38.53	5303.00	1.79
QUERY REQUESTS	0.00	0.00	0.00	0.00
CHANGE REQUESTS	3900.00	5.67	780.00	0.26
OTHER REQUESTS	0.00	0.00	0.00	0.00
LOCK ESCALATION (SHARED)	0.00	0.00	0.00	0.00
LOCK ESCALATION (EXCLUSIVE)	0.00	0.00	0.00	0.00
DRAIN REQUESTS	3.00	0.00	0.60	0.00
DRAIN REQUESTS FAILED	1.00	0.00	0.20	0.00
CLAIM REQUESTS	70227.00	102.05	14.0K	4.73
CLAIM REQUESTS FAILED	0.00	0.00	0.00	0.00

As we are interested in locking, all counters in this report are important to understand.

The first section is about *SUSPENSIONS*.

### ***SUSPENSIONS (ALL)***

This counter is sum of all the lock suspension counters that follow. The lower this number, the better. It should ideally be zero, but that is not very realistic.

### ***SUSPENSIONS (LOCK ONLY)***

The number of times an IRLM lock was requested but could not be granted right away. The lower this value, the better. When IRLM cannot grant a lock request immediately for whatever reason, and the requester asked to wait for it to become available (unconditional lock request), the lock request is suspended, and this counter is incremented.

### ***SUSPENSIONS (IRLM LATCH)***

These are suspensions within IRLM. To serialize resources inside IRLM, IRLM uses latches. When a transaction has to wait for an IRLM latch, this counter is incremented. High IRLM latch contention will drive up IRLM CPU time.



**Tip:** IRLM latch contention is considered high if the number of IRLM latch contentions is more than 1 to 5% of the number of IRLM requests. The number of IRLM requests is the sum of:

#LOCK REQUESTS + # UN LOCK REQUESTS + #CHANGE REQUESTS

From the values in Example 8-11 we have:

$0.55 / (758.37 + 38.53 + 5.67) * 100 = 0.06\%$

This is certainly an acceptable value.

One reason for high IRLM latch suspension is that IRLM is not running at the proper priority level in the system. The IRLM address space should use a service class that has a priority higher than the other DB2 system address spaces. The use of the WLM SYSSTC service class is recommended for IRLM.

An increase in IRLM latch contention can also occur when a lot of RELEASE(DEALLOCATE) plans and packages are used. Since they keep their page set level locks around for a longer time than RELEASE(COMMIT), this can increase the total number of concurrent locks to be managed by IRLM, making longer resource chains to process, which increases the time an IRLM latch is held and can cause contention with another requester during that time.

Another possible reason for high number of IRLM latch suspensions is when the deadlock detection cycle is very low, and the locking rate is very high. During deadlock detection, the IRLM main latch is on. If IRLM has to check for deadlocks very frequently, and the number of locks to check is very high, the main latch is held for a longer time and IRLM latch contention is more likely to occur.

### ***SUSPENSIONS (OTHER)***

This bucket collects the “none of the above” suspensions. The number of OTHER suspensions is expected to be low, except when DB2 sysplex parallelism is used. Types of request that go into this bucket are:

► IRLM SYNC requests

Those are issued when a DB2 agent abnormally ends and IRLM and DB2 need to be synced up again. You may expect up to one request per agent cancelled.

► IRLM synchronous notifies.

These can occur for a number of reasons, including IRLM synchronous notifies. Examples include:

- Requests to invalidate DBDs in other member
- Requests to extend a data set
- Requests for drop or revoke to invalidate EDM authorization caches in other members
- Sysplex query parallelism
- GBP recovery

The second section of the locking activity report describes the number of *timeouts* and *deadlocks*.

Both counters provide a good starting point for locking and concurrency analysis. Obviously, the objective is to keep the number of both as close to zero as possible. If there is a non-zero value for either of them, you need to investigate them. Section 9.2, “Analysis of a simple deadlock scenario” on page 280 provides a step by step procedure on how to analyze a deadlock, timeout, or locking problem in general.

The third section contains the *number of requests* from DB2 to IRLM using the following categories.

### **LOCK REQUESTS**

This counter represents the number of (L-)LOCK requests that was sent to IRLM.

### **UNLOCK REQUESTS**

This counter represents the number of (L-)UNLOCK requests that was sent to IRLM. Remember that a single UNLOCK request can release many locks in a single operation. For example, at commit time, DB2 issues an UNLOCK ANY request to release all locks that are no longer required.

If a program that is using isolation level CS is fetching from a read only cursor, DB2 tries to avoid taking locks as much as possible (lock avoidance). However, DB2 may have to acquire locks as it fetches rows from the cursor. If a lock was acquired and the cursor moves off the row/page, that lock is released by an UNLOCK request.

So, DB2 mainly issues UNLOCK requests for this type of cursor fetching (unlocking a row/page at a time) or at commit (unlocking them all).

**Tip:** To assess the effectiveness of DB2's data lock avoidance techniques at a high level, you can calculate #UNLOCK/COMMIT. If that value is greater than 5, DB2 lock avoidance is not effective.

In that case, you may want to ensure that the application uses the ISOLATION(CS) and CURRENTDATA(NO) BIND options.

In the example above, UNLOCK/COMMIT is 1.79, which is a good value. This is a very high level sanity check. If, for example, the application is very light and is only doing a few fetches, even if lock avoidance is not working at all, the ratio will still be low. Therefore, it is always important to check the overall transaction profile and not just blindly apply any rules of thumb.

"Information about DB2 lock avoidance" on page 276 provides more information about the types of trace records to gather when you want to discover whether lock avoidance is effective or not for a certain application.

A high number of lock/unlock requests/commits can also be the result of updating a compressed or variable length row. In the case where the updated row does not fit on the original page, DB2 does a lock/unlock of the row pointer.

Both can be eliminated by running a REORG utility against the object (table space or index).

### **CHANGE REQUESTS**

This counter represents the number of (L-)CHANGE requests that were sent to IRLM. This type of request is normally used to "upgrade" an existing lock from one state to another, for example, to change a U lock that was acquired by an updateable cursor at fetch time, to an X lock when the row is actually updated by a positioned update.

The fourth section of the report contains information about *lock escalations* that occurred in the system during the reported time interval. DB2 statistics distinguish between lock escalations to shared *LOCK ESCALATION (SHARED)* and escalations to exclusive mode *LOCK ESCALATION (EXCLUSIVE)*. Both counters provide a good starting point for further locking and concurrency analysis. Obviously, the objective is to keep the number of both as close to zero as possible. If there is a non-zero value for either of them, you need investigate why they occur.

The fifth and last part of the locking activity section has information about *claims and drains*. Again, you do not want to see a non-zero value for *DRAIN REQUESTS FAILED* or *CLAIM REQUESTS FAILED*. Claim and drain failures or timeouts typically do not produce the regular deadlock or timeout messages, but normally a DSNT500I message with the resource name and a specific reason code will be written to the console.

Example 8-12 shows an example of a DSNT500I message from a utility that unsuccessfully tried to drain an object. The utility was unable to drain the object because a program (without commits) was holding on to a claim for a long period of time.

*Example 8-12 DSNT500I message*

---

```

DSNT500I  -DB9A DSNUGRAR RESOURCE UNAVAILABLE
          REASON 00C200EA
          TYPE 00000210
          NAME DBBART1 .TSPTMP .00000001

```

where:

#### **00C200EA**

Explanation: DB2 is unable to perform the drain function on an object (a table space, an index space, or a partition) because the object was held by other claimers and the drain request timed out waiting for the claim count to reach zero.

---

## **Statistics CPU times**

Example 8-13 shows the CPU times section of the DB2 statistics report. It indicates how much CPU time was used by the DB2 system address spaces, including IRLM.

*Example 8-13 Statistics CPU times*

CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	TOTAL TIME	PREEMPT IIP SRB	/COMMIT
-----	-----	-----	-----	-----	-----	-----
SYSTEM SERVICES ADDRESS SPACE	0.341285	0.000000	0.262623	0.603908	N/A	0.000041
DATABASE SERVICES ADDRESS SPACE	0.029325	0.000000	0.320044	0.349369	0.000000	0.000024
<b>IRLM</b>	<b>0.000020</b>	<b>0.000000</b>	<b>1.786929</b>	<b>1.786949</b>	<b>N/A</b>	<b>0.000120</b>
DDF ADDRESS SPACE	0.152151	14.353293	0.442421	14.947865	14.833588	0.001007
TOTAL	0.522781	14.353293	2.812018	17.688091	14.833588	0.001191

---

As you can see, almost all IRLM CPU time is non-preemptible SRB time. This is expected, as most IRLM processes run as an SRB. In our example, the IRLM CPU time is about 120 microseconds per commit, which is very reasonable.

IRLM CPU usage is normally low compared to MSTR/DBM1 CPU usage.

The major contributors to IRLM CPU usage are:

- ▶ Lock resume processing (of a lock request that was suspended because the lock was not available when it was initially requested).
- ▶ Deadlock detection processing.
- ▶ Local IRLM latch contention processing.
- ▶ Activating detailed IRLM component traces.

In a data sharing environment (refer to Part 4, “Data sharing” on page 303), there are a number of additional processes in IRLM that can consume CPU cycles, such as:

- ▶ IRLM and XES global contention / notify processing
- ▶ Asynchronous XES request handling
- ▶ P-lock negotiation
- ▶ Asynchronous child lock propagation
- ▶ Global deadlock processing
- ▶ IRLM XCF group heartbeat monitoring

### Statistics DB2 latch contention per second

The DB2 statistics record also has a section that shows the number of DB2 latch contentions per second during the interval that is covered by the report. Note that these counters are only incremented when *latch contention* occurs. Acquiring a latch without contention does not increase these counters. For a general explanation about the different DB2 latch classes, refer to 3.8, “Latches” on page 56.

Each number in the data shown in Example 8-14 is for one of the 33 (32+1) latch classes that are used by DB2. The first row is for latch class 1(LC01), LC02, LC03 and LC04, and so on.

*Example 8-14 DB2 latch contention per second*

LATCH CNT	/SECOND	/SECOND	/SECOND	/SECOND
-----	-----	-----	-----	-----
LC01-LC04	0.00	0.00	0.00	0.00
LC05-LC08	0.00	0.00	0.00	0.00
LC09-LC12	0.00	0.07	0.00	0.08
LC13-LC16	0.00	37.56	0.00	0.00
LC17-LC20	0.00	0.00	0.46	0.00
LC21-LC24	0.00	0.00	0.35	11613.47
LC25-LC28	0.25	0.00	0.38	0.00
LC29-LC32	0.00	0.08	0.16	4.47
LC254	0.00			

DB2 systems will always have some degree of latch contention. This is inherent to the way latches are used by the DB2 system. Therefore, a non-zero value does not necessarily mean there is a problem.

#### Tip:

- ▶ A latch contention rate of less than 1 K per second for a DB2 latch class is not a problem.
- ▶ A latch contention rate of less than 1 K to 10 K per second for a certain latch class means you are getting into the danger zone.
- ▶ A latch contention rate or greater than 10K per second often means you are hitting some sort of bottleneck in the system that needs to be investigated.

Each *DB2 latch class* usually represents more than one *DB2 latch*. All individual DB2 latches are mapped to the 32 +1 latch classes that you see in the DB2 statistics report. Most of the time, just looking at the latch class is enough to understand which one of the underlying individual latches is causing the problem. If that is not the case, you can trace IFCID 56 (exclusive latch wait) and IFCID 57 (exclusive latch resume). These IFCIDs have the real DB2 latch number and the address of the latch. The following trace command will gather this information:

```
-STA TRA(P) CLASS(30) IFCID(56,57) DEST(SMF/GTF) TDATA(CPU COR DIST)
```

Example 8-14 on page 234 shows that the latch contention rate for LC24 is 11613.47 per second. Based on the rules of thumb above, this value is in the danger zone, and needs to be investigated in more detail.

Note that activating these IFCIDs produces a large number of trace records. Therefore, you typically only turn on this trace for one minute (or even for 15 seconds) on a very busy system. As the trace volume is large, and to avoid flooding SMF and losing some SMF records that are used for charge back, GTF is usually preferred as a trace destination for high volume tracing.

Latches are described in 3.8, “Latches” on page 56.

DB2 accounting records also contain information about DB2 latch suspensions incurred by transactions when DB2 accounting trace class 3/8 is active. Gathering this information (counting the number of DB2 latch suspensions and the time of each suspension) can be costly when the system shows a very large number of DB2 latch contentions per second. Disabling DB2 accounting class 3 can help reduce CPU time when very high latch contention is occurring.

## Statistics SQL DCL section

The DB2 statistics SQL DCL section, among a number of other things, tracks the number of LOCK TABLE statements that were issued by all applications in the subsystem during the interval the DB2 statistics report is covering.

Example 8-15 shows the SQL DCL section of the DB2 statistics report.

*Example 8-15 SQL DCL*

SQL DCL	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
<b>LOCK TABLE</b>	0.00	0.00	0.00	0.00
GRANT	0.00	0.00	0.00	0.00
REVOKE	0.00	0.00	0.00	0.00
SET HOST VARIABLE	0.00	0.00	0.00	0.00
SET CURRENT SQLID	0.00	0.00	0.00	0.00
SET CURRENT DEGREE	0.00	0.00	0.00	0.00
...				

Large numbers for the LOCK TABLE values may have a negative effect on concurrency. To determine whether this is the case, you can check the lock suspension, timeout, and deadlock counters in the LOCKING ACTIVITY (refer to “Statistics locking activity” on page 230).

### 8.3.3 Analyzing DB2 accounting data

In the previous section, we looked at the locking information at the subsystem level using the DB2 statistics report.

To get an idea whether, and if so, which applications suffer from concurrency problems or have been issued a large number of lock requests, we use the DB2 accounting information.

The OMEGAMON PE ACCOUNTING REPORT report calculates average times. These averages are based on the number of occurrences. The data presented in Example 8-16 indicates that the number of occurrence is 93. It also shows that no accounting rollup (<sup>2</sup>=NO) is being performed, because the #DDFRRSAF ROLLUP counter is zero. If no rollup accounting is performed, #OCCURRENCES is equal to the number of DB2 accounting records that were encountered in the reporting interval.

*Example 8-16 Accounting highlights information*

---

HIGHLIGHTS

-----

#OCCURRENCES	:	93
#ALLIEDS	:	0
#ALLIEDS DISTRIB:		0
#DBATS	:	93
#DBATS DISTRIB. :		0
#NO PROGRAM DATA:		0
#NORMAL TERMINAT:		93
#DDFRRSAF ROLLUP:		0
#ABNORMAL TERMIN:		0
#CP/X PARALLEL. :		0
#IO PARALLELISM :		0
#INCREMENT. BIND:		0
#COMMIT	:	86
#ROLLBACKS	:	7
#SVPT REQUESTS	:	0
#SVPT RELEASE	:	0
#SVPT ROLLBACK	:	0
MAX SQL CASC LVL:		1
UPDATE/COMMIT	:	3.26
SYNCH I/O AVG.	:	0.001069

---

If you are using ACCUMACC, #OCCURRENCES is the total number of transactions. For example, if ACCUMACC=10 and you have 20 rollup accounting records (#DDFRRSAF ROLLUP = 20), the number of occurrences will be 200.

---

<sup>2</sup> The DB2 subsystem parameter ACCUMACC controls whether and when DB2 accounting data is accumulated by the user for DDF and RRSF threads.

## Accounting class 1,2,3 times

When analyzing DB2 accounting data, you usually start by looking at the elapsed time distribution (see Example 8-17) of the transactions. This is an easy way to determine whether the problem is in the application or in DB2, and when it is in DB2, whether it is a CPU time (high CL2 CPU) problem, a DB2 suspension problem (CL3 time), or most likely a system problem (high NOTACC time).

In this case, it is clearly a DB2 suspension problem.

*Example 8-17 Elapsed time distribution*

ELAPSED TIME DISTRIBUTION		CLASS 2 TIME DISTRIBUTION	
APPL	=> 4%	CPU	=> 2%
DB2	=> 3%	SECPU	
SUSP	===== 93%	NOTACC	> 1%
		SUSP	===== 97%

The next thing to do is to see which type of suspension is causing most of the delays. Example 8-18 shows the accounting class 1,2,3 times of the accounting report.

*Example 8-18 Accounting class 1,2,3 times*

AVERAGE	APPL(CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	AVERAGE TIME	AV.EVENT
ELAPSED TIME	1.851638	1.778867	N/P	<b>LOCK/LATCH(DB2+IRLM)</b>	<b>1.716900</b>	<b>21.53</b>
NONNESTED	0.061021	0.010285	N/A	SYNCHRON. I/O	0.003310	3.10
STORED PROC	1.790617	1.768582	N/A	DATABASE I/O	0.002357	2.83
UDF	0.000000	0.000000	N/A	LOG WRITE I/O	0.000953	0.27
TRIGGER	0.000000	0.000000	N/A	OTHER READ I/O	0.007764	2.25
				OTHER WRTE I/O	0.000000	0.00
CP CPU TIME	0.046463	0.031898	N/P	SER.TASK SWTCH	0.003052	0.40
AGENT	0.046463	0.031898	N/A	UPDATE COMMIT	0.001908	0.08
NONNESTED	0.006549	0.005566	N/P	OPEN/CLOSE	0.000000	0.00
STORED PRC	0.039914	0.026331	N/A	SYSLGRNG REC	0.000523	0.11
UDF	0.000000	0.000000	N/A	EXT/DEL/DEF	0.000000	0.00
TRIGGER	0.000000	0.000000	N/A	OTHER SERVICE	0.000620	0.22
PAR.TASKS	0.000000	0.000000	N/A	ARC.LOG(QUIES)	0.000000	0.00
				LOG READ	0.000000	0.00
SECP CPU	0.003632	N/A	N/A	<b>DRAIN LOCK</b>	0.000000	0.00
				<b>CLAIM RELEASE</b>	0.000000	0.00
SE CPU TIME	0.000000	0.000000	N/A	<b>PAGE LATCH</b>	0.000021	0.09
NONNESTED	0.000000	0.000000	N/A	<b>NOTIFY MSGS</b>	0.000000	0.00
STORED PROC	0.000000	0.000000	N/A	<b>GLOBAL CONTENTION</b>	0.000000	0.00
UDF	0.000000	0.000000	N/A	COMMIT PH1 WRITE I/O	0.000000	0.00
TRIGGER	0.000000	0.000000	N/A	ASYNCH CF REQUESTS	0.000000	0.00
				TCP/IP LOB	0.000031	0.66
PAR.TASKS	0.000000	0.000000	N/A	TOTAL CLASS 3	1.731078	28.01
SUSPEND TIME	0.001687	1.731078	N/A			
AGENT	N/A	1.731078	N/A			
PAR.TASKS	N/A	0.000000	N/A			
STORED PROC	0.001687	N/A	N/A			
UDF	0.000000	N/A	N/A			
NOT ACCOUNT.	N/A	0.015891	N/A			
DB2 ENT/EXIT	N/A	13.94	N/A			
EN/EX-STPROC	N/A	199.91	N/A			
EN/EX-UDF	N/A	0.00	N/A			
DCAPT.DESCR.	N/A	N/A	N/P			
LOG EXTRACT.	N/A	N/A	N/P			

The bold fields are the ones that are related to locking and serialization. As this is our primary focus, we discuss them in detail hereafter.

All the class 3 fields in the OMEGAMON PE accounting report give the average time that the (average) transaction is suspended, but also give the (average) number of times the suspension occurs. In other words, you want to know whether you are dealing with a few long suspensions, or many short suspensions.

### **LOCK/LATCH(DB2+IRLM)**

Different types of suspensions go into this bucket, so it is not always easy to determine which one is really making up the bulk of the suspensions or suspension time. The three contributors to LOCK/LATCH(DB2+IRLM) suspension are:

- **DB2 local lock contention**

This is the time a transaction is waiting for a local lock (held by another transaction on this subsystem or member). In the vast majority of the cases, this will be the main contributor to the LOCK/LATCH wait time counter.

- **DB2 latch contention**

This is the time a transaction has to wait for a DB2 latch to become available. Latches are normally held for a very short time, but if the DB2 statistics report shows a large number of latch waits per second (refer to “Statistics DB2 latch contention per second” on page 234), DB2 latch wait time can become a significant contributor to the LOCK/LATCH wait time counter.

- **IRLM latch contention**

This is the time a transaction was processing an IRLM request but was suspended waiting for a latch in IRLM to become available. If the DB2 accounting locking section (refer to “Accounting locking” on page 241) shows a large number of IRLM latch suspensions, IRLM latch suspension wait time can become a significant contributor to the LOCK/LATCH wait time counter.

### **DRAIN LOCK**

This the time the transaction has been waiting for a drain lock to become available. Drain locks are usually acquired by utilities. Therefore, when there is a high drain wait time, it is worthwhile checking whether there were any utilities running concurrently with the transactions.

You can also check the Accounting DRAIN/CLAIM section (refer to “Accounting drain and claim” on page 242) to verify the number of drains that were issued by the job or transaction.

### **CLAIM RELEASE**

This the time the transaction or job has been waiting for a claim to be released. Claims are normally released at commit time (a noticeable exception being open cursors with hold). Therefore, if a job or transaction has to wait for a long time for a claim to be released, it is most likely because another job or transaction in the system is not committing frequently. If you suspect that is the case, it is worthwhile to check the DB2 console messages in the MSTR joblog or syslog for warnings about long running units of work (refer to 8.2.2, “Console messages” on page 222).

A utility that places a drain lock has to wait for the claimers to release their claim (at commit), so it is likely that has a high CLAIM RELEASE wait time is going to be one of the DB2 utilities.



The difference between a DRAIN LOCK wait and CLAIM RELEASE wait is that, when suspended for a DRAIN LOCK, the application was suspended when it issued the lock request to obtain the drain lock; that is, another application is holding the drain lock in an incompatible state. Waiting for CLAIM RELEASE happens after the drain lock has been obtained. Once the drain lock has been acquired, if there are still claimers for the resource, the application waits for all the claimers to go away before the drain is considered successful. This wait time is recorded as CLAIM RELEASE wait time.

Both waits typically occur for utilities, but these wait times can also show up for normal applications, for example, when the first claim on an object is acquired, and a utility is still active against the object (holding a drain lock).

Refer to 3.5, “Claims and drains” on page 53 for the definition of these functions.

### ***Page latch wait time***

Page latch wait times are often a symptom of some other type of contention.

When a page is being accessed, DB2 acquires a latch on a page, which prevents a transaction from seeing a page in an inconsistent state. For example, when DB2 is in the process of inserting a row into a page, you do not want any other transactions to see the page in an intermediate state. If the state that you want to acquire the latch at is incompatible with an existing latch, the transaction is suspended and will have to wait for the page latch to become available.

Another example is that when a write I/O is in progress for a page, the page is not allowed to be updated. A transaction that wants to access this page while it is being written out is suspended until the write I/O is completed. This wait event and its associated wait time is captured either in the “other write I/O”, or in the “page latch” category. For the first updating transaction, this wait time is captured in “other write I/O”. All other subsequent update transactions’ wait time are captured as part of “page latch” wait time. So, as indicated above, page latch contention is often a symptom of something else; in this case, slow write I/Os. Another case can be when a set of pages that are frequently updated are constantly being written out (over and over again) because of a low vertical deferred write threshold repetitively flushing the buffers.

In a data sharing environment with row level locking, when multiple threads are inserting to or updating the same page, the first thread acquires a page P-lock, and the suspension for other threads is reported as page latch suspensions. The same is true for page P-locks on index leaf pages and spacemap pages.

If it is not clear from the plan or package information which objects are causing the high page latch contention, you can analyze page latch contention reports based on IFCID 226 and IFCID 227, which report each individual page latch contention event, provide their duration, and identify the object(s) of contention. The trace is started with the following command:

```
-STA TRA(P) CLASS(30) IFCID(226,227) DEST(SMF/GTF) TDATA(CPU COR DIST)
```

This trace is likely to produce a very high volume of trace data, so do not run it for a long time; 30 seconds to 1 minute during a peak contention period is normally enough.

The tuning actions to reduce the amount of page latch contention depend on what the underlying root cause is and the type of pages the page latch contention is for.

If the contention is on frequently updated pages that are written back to disk all the time, a tuning option is to keep them in the buffer pool as long as possible without writing them out by increasing the deferred write thresholds.

For many occasions, the contention is on table space “space map” pages. To relieve space map contention on table spaces, you can partition the object so that you have more space map pages, or change the clustering index in a way that the inserts are spread out across more space map pages.

In a data sharing environment, using TRACKMOD NO or the MEMBER CLUSTER table space attribute can help cool off space map pages. However, you have to keep in mind that using MEMBER CLUSTER can also lead to more page latch contention when used unwisely. In a MEMBER CLUSTER table space, a space map page only covers 199 pages, so there will be many more space map pages in the table space to latch, for example, during an exhaustive space search.

Note that page latches are not part of any of the 32+1 DB2 latch classes we discussed before, nor do page latches result in a request to IRLM.

### ***Notify messages wait time***

Waiting for notify message reply time only applies to a data sharing environment. This time is the (average) accumulated elapsed wait time caused by suspension for sending messages to other members in the data sharing group. These *NOTIFY* messages are processed by way of IRLM. One common use of inter-system message sending is when database descriptors (DBDs) are changed due to CREATE, ALTER, or DROP statements or when objects have to be drained across the data sharing group.

### ***Global contention wait time***

This type of class 3 suspension also occurs only in a data sharing environment. When you see a large number of global contention wait times, or a high number of global contention wait events, refer to the global contention section in the accounting report.

Example 8-19 shows the report with a breakdown of the global contention counters in global L-lock versus P-lock wait time, and per type whether the suspensions are at the table space / partition, or the page/row level. The data in Example 8-19 is from a non-data sharing system, so the values are obviously all zero. We just use this example to illustrate that the information about global lock contention in the DB2 accounting record is very granular.

*Example 8-19 Global contention L-locks and P-locks*

GLOBAL	CONTENTION	L-LOCKS	AVERAGE TIME	AV.EVENT	GLOBAL	CONTENTION	P-LOCKS	AVERAGE TIME	AV.EVENT
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
L-LOCKS			0.000000	0.00	P-LOCKS			0.000000	0.00
PARENT (DB,TS,TAB,PART)			0.000000	0.00	PAGESET/PARTITION			0.000000	0.00
CHILD (PAGE,ROW)			0.000000	0.00	PAGE			0.000000	0.00
OTHER			0.000000	0.00	OTHER			0.000000	0.00

## Accounting locking

After looking at the accounting class 1,2,3 times, we discover another important section, that is, the accounting LOCKING section (refer to Example 8-20).

*Example 8-20 Accounting locking information*

LOCKING	AVERAGE	TOTAL
-----	-----	-----
TIMEOUTS	0.05	5
DEADLOCKS	0.05	5
ESCAL.(SHARED)	0.00	0
ESCAL.(EXCLUS)	0.00	0
<b>MAX PG/ROW LOCKS HELD</b>	2.62	13
LOCK REQUEST	81.20	7552
UNLOCK REQUEST	16.62	1546
QUERY REQUEST	0.00	0
CHANGE REQUEST	7.70	716
OTHER REQUEST	0.00	0
TOTAL SUSPENSIONS	0.39	36
LOCK SUSPENSIONS	0.29	27
IRLM LATCH SUSPENS.	0.10	9
OTHER SUSPENS.	0.00	0

The information in the DB2 accounting locking section is very similar to the information in the DB2 statistics locking section (refer to “Statistics locking activity” on page 230), except for the claim and drain information that is in a separate section in the DB2 accounting report (refer to “Accounting drain and claim” on page 242). However, the DB2 accounting locking information is for a specific transaction (or group of transactions, depending on the reporting criteria), while the locking information in the DB2 statistics report is for the entire DB2 subsystem or data sharing member.

### **MAX PG/ROW LOCKS HELD**

There is one interesting piece of information in the accounting locking section that is not available in the statistics record. This is the *MAX PG/ROW LOCKS HELD* field. It is a useful indicator of the application’s commit frequency. The counter applies to low level locks only (page or row, LOB and XML). The value cannot exceed the maximum number of locks per user that is allowed by the DSNZPARM NUMLKUS. If you exceed NUMLKUS, the next lock request will result in a “RESOURCE UNAVAILABLE” message with a 00C90096 reason code.

**Tip:** In general, try to issue a COMMIT frequently enough to keep the average MAX PG/ROW LOCKS HELD below 100.

Note that the AVERAGE value that is shown in the accounting REPORT is the average of MAX PG/ROW LOCKS HELD of all the accounting records that qualify for the report. The TOTAL is for the maximum of all MAX PG/ROWS LOCKS HELD, that is, the “high water mark” of all accounting records that qualify for the report.

For example, if transaction A has a MAX PG/ROWS LOCKS HELD value of 10, and transaction B has a MAX PG/ROWS LOCKS HELD value of 20, then an accounting report that includes these two transactions will have AVERAGE (average of max.) of 15, and TOTAL (high water mark) of 20.

## Accounting drain and claim

Example 8-21 provides information about claim and drain requests, as well as the number of drain and claim requests that failed. This information is similar to the counters in the DB2 statistics locking section. For a more detailed description of the counters, refer to Example 8-11 on page 230. The difference is that in the statistics record, these counters are at the subsystem or DB2 member level, while in the accounting report, they are at the transaction level (or at whatever other level the accounting data is summarized).

For a general description of claims and drains, refer to 3.5, “Claims and drains” on page 53.

*Example 8-21 Accounting DRAIN/CLAIM information*

DRAIN/CLAIM	AVERAGE	TOTAL
-----	-----	-----
DRAIN REQUESTS	0.00	0
DRAIN FAILED	0.00	0
CLAIM REQUESTS	39.42	3666
CLAIM FAILED	0.00	0

## Accounting SQL DCL

The DB2 accounting SQL DCL section, among a number of other things, tracks the number of LOCK TABLE statements that were issued by the applications that are included in the DB2 accounting report interval.

Example 8-22 shows the SQL DCL section of the DB2 accounting report.

*Example 8-22 Accounting SQL DCL information*

SQL DCL	TOTAL
-----	-----
<b>LOCK TABLE</b>	0
GRANT	0
REVOKE	0
SET CURR.SQLID	0
SET HOST VAR.	51
SET CUR.DEGREE	0
...	

Applications that issue a LOCK TABLE request can avoid having to take a large number of lower level locks (page or row locks) and reduce the CPU usage of the job or transaction, but it will be at the expense of concurrency if other applications try to access the object that is locked with a table/table space lock at the same time. It is therefore a good idea to have a quick look at the number of LOCK TABLE requests that are issued by the transactions in your system. This is often the next logical step when the DB2 statistics have indicated that LOCK TABLE statements have been issued. You can then look at the DB2 accounting information, for example, at the plan level, to identify which plans issued the LOCK TABLE statements. This counter shows the executions through the SQL interface of the LOCK TABLE statement, and is not related to the number of table locks that are acquired by DB2, such as for lock escalation.

## 8.3.4 Periodic monitoring of DB2 system level concurrency related IFCIDs

In this section, we take a closer look at the following IFCIDs:

- ▶ IFCID 172: deadlock information
- ▶ IFCID196: timeout information
- ▶ IFCID 313: long running units of recovery
- ▶ IFCID 337: lock escalation

They are all created if DB2 statistics trace class 3 is active. Collecting this information has a very low impact and the information is very valuable when analyzing concurrency problems.

### IFCID 172: deadlock information

The easiest way to generate a batch report that formats IFCID 172 is to use the following OMEGAMON PE command:

```
LOCKING
TRACE
LEVEL(LOCKOUT)
```

An excerpt of the report is shown in Example 8-23.

**Note:** The advantage of using IFCID 172 to analyze a deadlock is that IFCID 172 has information about all the resources and parties involved. The DSNT375I and DSNT501I messages only contain information about one of the resources and users involved.

To analyze a deadlock, you have to understand which transactions and which resources are involved.

Example 8-23 IFCID 172 in a LOCKOUT trace

```
LOCATION: DB9A                                OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)                                PAGE: 1-2
GROUP: N/P                                  LOCKING TRACE - LOCKOUT                                REQUESTED FROM: NOT SPECIFIED
MEMBER: N/P                                ACTUAL FROM: 07/28/09 06:34:12.13
SUBSYSTEM: DB9A                                PAGE DATE: 07/28/09
DB2 VERSION: V9                                SCOPE: MEMBER
PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE
PLANNAME CONNECT
-----
BART      BARTSQR3 DB2CALL      06:37:47.25758212 DEADLOCK
BART      'BLANK'   C48CE96572CA N/P
DSNREXX   DB2CALL
                                     DATAPAGE DB  =GLWBSQLP
                                     OB   =GLWSEMP
                                     PAGE=X'00010B'
                                     -----
                                     COUNTER =25856    WAITERS = 2
                                     TSTAMP  =07/28/09 06:37:47.25
                                     HASH    =X'0048490B'
                                     ----- BLOCKER is HOLDER --*VICTIM*-
                                     LUW=USIBMSC.SCPDB9A.C48CE96572CA
                                     MEMBER  =N/P      CONNECT =DB2CALL
                                     PLANNAME=DSNREXX  CORRID  =BARTSQR3
                                     DURATION=COMMIT   PRIMAUTH=BART
                                     STATE   =X
                                     PROGNAME=DEDLCK
                                     COLLID  =GLWBSQLP
                                     LOCATION=N/P
                                     CONTOKEN=X'18907CB5106DCC28'
                                     ----- WAITER -----
                                     LUW=USIBMSC.SCPDB9A.C48CE974E055
                                     MEMBER  =N/P      CONNECT =DB2CALL
                                     PLANNAME=DSNREXX  CORRID  =BARTSQR4
                                     DURATION=COMMIT   PRIMAUTH=BART
                                     REQUEST =LOCK      WORTH   = 18
                                     STATE   =X
                                     PROGNAME=DEDLCK
                                     COLLID  =GLWBSQLP
                                     LOCATION=N/P
                                     CONTOKEN=X'18907CB5106DCC28'
                                     -----
                                     DATAPAGE DB  =GLWBSQLP
                                     OB   =3
                                     PAGE=X'000002'
                                     HASH    =X'000B8802'
                                     ----- BLOCKER is HOLDER -----
                                     LUW=USIBMSC.SCPDB9A.C48CE974E055
                                     MEMBER  =N/P      CONNECT =DB2CALL
```

```

PLANNAME=DSNREXX  CORRID =BARTSQR4
DURATION=COMMIT   PRMAUTH=BART
STATE =X
PROGNAME=DEDLCK
COLLID =GLWBSQLP
LOCATION=N/P
CONTOKEN=X'18907CB5106DCC28'
----- WAITER -----*VICTIM*-
LUW=USIBMSC.SCPDB9A.C48CE96572CA
MEMBER =N/P      CONNECT =DB2CALL
PLANNAME=DSNREXX CORRID =BARTSQR3
DURATION=COMMIT   PRMAUTH=BART
REQUEST =LOCK     WORTH  = 17
STATE =X
PROGNAME=DEDLCK
COLLID =GLWBSQLP
LOCATION=N/P
CONTOKEN=X'18907CB5106DCC28'

```

---

The victim of the deadlock is the transaction that has to give up its locks, allowing the other transaction(s) to continue. In the OMEGAMON PE lockout trace, this is always the thread on the left hand side of the report, in this case, job BARTSQR3 (instance C48CE96572CA). On the right hand side, this thread will be marked as *\*VICTIM\**. Note that the victim will be holding some resources while waiting for others.

The other transactions that are involved in the deadlock show up on the right hand side either as a blocker or a holder of a resource. In this case, there is only one other job involved and that is BARTSQR4 (instance C48CE974E055).

Note that the deadlock record not only contains the plan name, correlation-ID, and authid information, but also the member name (if it is a global deadlock), the collection-ID, and the program/package name. This should allow you to easily identify which programs need to be investigated to avoid the deadlock.

The resources that are involved in the deadlock are listed in the middle of the report, under the -- LOCK RESOURCE -- heading. In this case there are two resources involved:

```

DATAPAGE DB = GLWBSQLP OB = GLWSEMP    PAGE=X'00010B'
DATAPAGE DB = GLWBSQLP OB = 3          PAGE=X'000002'

```

Note that the OBID (OB) of the second resource is a (decimal) number. This normally happens when the object is segmented. The number is the OBID of the table. To verify, you can do a quick query against the DB2 catalog, as shown in Example 8-24.

*Example 8-24 Retrieving table OBID information from the catalog*

```

SELECT SUBSTR(CREATOR,1,8) AS CREATOR
      , SUBSTR(NAME,1,25) AS NAME
      , OBID
      , DBNAME
      , TSNAME
FROM SYSIBM.SYSTABLES WHERE DBNAME = 'GLWBSQLP'
AND TYPE = 'T'
AND OBID = 3
ORDER BY OBID;

```

CREATOR	NAME	OBID	DBNAME	TSNAME
GLWBSQLP	GLWTDPT	3	GLWBSQLP	GLWSDPT

So now that you know the transactions and resources that are involved, you need to understand how they lead up to the deadlock. This is shown in Figure 8-1.

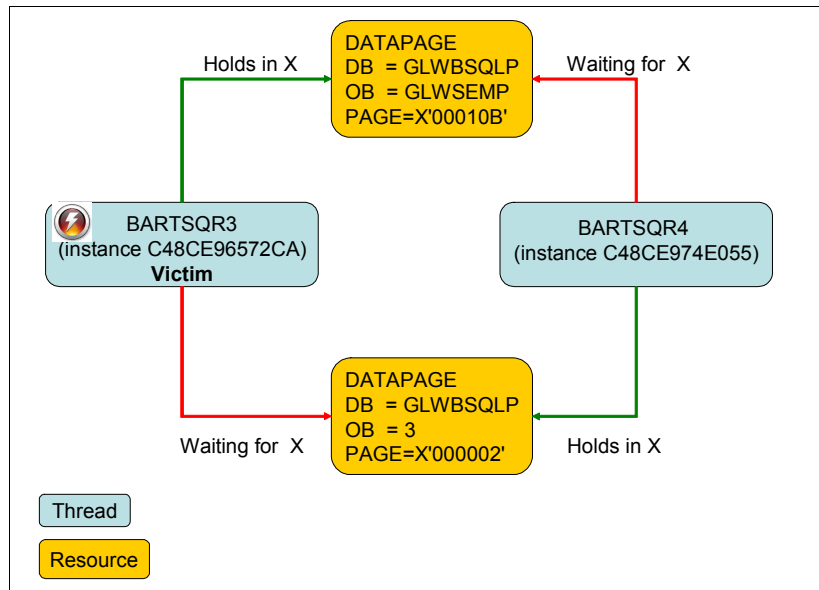


Figure 8-1 Threads and resources involved in a deadlock

As you can see, BARTSQR3 is holding an X lock on page x'10B' of GLWSEMP, and is waiting for an X lock on page x'2' of GLWDPT (OB = 3).

BARTSQR4 is doing the exact opposite. It is waiting for an X lock on page x'10B' of GLWSEMP, while holding an X lock on page x'2' of GLWDPT, so this is a classic example of a deadlock.

In this section, we only look at the deadlock record itself and how to interpret the information that it contains. Sections 9.2, "Analysis of a simple deadlock scenario" on page 280 and 5.19, "How to prevent locking problems" on page 150 describe how to analyze, in more detail, what leads up to the deadlock and how to avoid running into a deadlock situation.

## IFCID196: timeout information

Timeout records are much simpler to analyze than deadlock records. Example 8-25 shows the output of an OMEGAMON PE lockout trace that contains a timeout record.

As with the deadlock record, the thread that is timed out is presented on the left hand side of the report, in this case, jobname BARTSQR1 (instance C48CE87A4B7F).

*Example 8-25 IFCID 196 in a LOCKOUT trace*

LOCATION: DB9A		OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)				PAGE: 1-1	
GROUP: N/P		LOCKING TRACE - LOCKOUT				REQUESTED FROM: NOT SPECIFIED	
MEMBER: N/P						TO: NOT SPECIFIED	
SUBSYSTEM: DB9A						ACTUAL FROM: 07/28/09 06:34:12.13	
DB2 VERSION: V9		SCOPE: MEMBER				PAGE DATE: 07/28/09	
PRIMAUTH	CORRNAME	CONNTYPE					
ORIGAUTH	CORRNMBR	INSTANCE	EVENT	TIMESTAMP	---	LOCK	RESOURCE ---
PLANNAME	CONNECT		RELATED	TIMESTAMP	EVENT	TYPE	NAME
-----							
BART	BARTSQR1	DB2CALL	06:34:12.13277590	TIMEOUT	TABLE	DB	=754
BART	'BLANK'	C48CE87A4B7F	N/P			OB	=3
DSNREXX	DB2CALL						
-----							
						REQUEST	=LOCK
						STATE	=IS
						ZPARM	INTERVAL= 60
						DURATION	=COMMIT
						INTERV.COUNTER	= 1
						HASH	=X'000203F2'
						-----	HOLDERS/WAITERS -----
HOLDER							
LUW=USIBMSC.SCPDB9A.C48CE88CCBAD							
MEMBER =N/P							
CONNECT =DB2CALL							
PLANNAME=DSNREXX							
CORRID =BARTSQR2							
DURATION=COMMIT							
PRIMAUTH=BART							
STATE =X							

The resource that DB2 timed out on is in the middle under the -- LOCK RESOURCE -- heading. In this case, both the database and table / table space are shown using their OBID number, not their name. So, again, we can SELECT FROM the catalog to retrieve the name (refer to Example 8-26). As the type indicates that we timed out on a TABLE lock, we can assume that OBID 3 is a table identifier, so we check in SYSIBM.SYSTABLES. If the OBID is not found there, you can check SYSTABLESPACE and use the PSID.

Note that timeout interval is 60 seconds on this subsystem.

*Example 8-26 Retrieving table OBID information from the catalog - 2*

```
SELECT SUBSTR(CREATOR,1,8) AS CREATOR
      , SUBSTR(NAME,1,25) AS NAME
      , SUBSTR(DBNAME,1,8) AS DBNAME
      , SUBSTR(TSNAME,1,8) AS TSNAME
      , DBID
      , OBID
FROM SYSIBM.SYSTABLES WHERE
TYPE = 'T'
AND DBID = 754
AND OBID = 3 ;
```

CREATOR	NAME	DBNAME	TSNAME	DBID	OBID
GLWBSQLP	GLWTDPT	GLWBSQLP	GLWSDPT	754	3

In case of a timeout, it is the waiter that times out after the timeout interval has expired. The holder just continues to hold its lock. In Example 8-25, the holder is job (corrid) BARTSQR2 (instance C48CE88CCBAD) and it is holding the table lock in X mode. The waiter, that timed out, requested the table lock in IS mode. In this case, job BARTSQR2 issued the following SQL statement:

```
LOCK TABLE GLWBSQLP.GLWTDPT IN EXCLUSIVE MODE
```



It issued this SQL statement before it started its batch update process. This reduces the number of locks that BARTSQR2 has to acquire, but as you can see, it also reduces the concurrency and caused another job, BARTSQR1, that only wants to read the table (hence its IS lock request), to time out.

In this example, there is only one waiter, and that is the thread that timed out. It is not unusual to see more than one waiter for a resource in a timeout record. Only the thread that timed out stops waiting for the lock. The other waiters remain waiting for the lock to become available. If they also wait longer than the timeout value, they receive a timeout at that time. There can also be multiple holders of a lock.

The advantage of analyzing IFCID 196 is that it contains all holders and waiters, while the DSNT376I only contains information about the thread that timed out and one holder of the resource.

Increasing the commit frequency of BARTSQR2 to a value less than the timeout interval should prevent BARTSQR1 from timing out. But even though it may prevent BARTSQR1 from timing out, it may still cause relatively long lock suspension wait times for this job.

Example 8-27 is another timeout record, but in this case we time out because of an X lock on a data page.

Example 8-27 IFCID 196 in a LOCKOUT trace - 2

LOCATION: DB9A			OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)			PAGE: 1-3		
GROUP: N/P			LOCKING TRACE - LOCKOUT			REQUESTED FROM: NOT SPECIFIED		
MEMBER: N/P						TO: NOT SPECIFIED		
SUBSYSTEM: DB9A						ACTUAL FROM: 07/28/09 06:34:12.13		
DB2 VERSION: V9			SCOPE: MEMBER			PAGE DATE: 07/28/09		
PRIMAUTH	CORRNAME	CONNTYPE	EVENT TIMESTAMP	--- L O C K R E S O U R C E ---				
ORIGAUTH	CORRNMBR	INSTANCE	RELATED TIMESTAMP	EVENT	TYPE	NAME	EVENT SPECIFIC DATA	
PLANNAME	CONNECT							
BART	BARTSQR3	DB2CALL	06:38:19.30855822	TIMEOUT	DATA PAGE	DB =GLWBSQLP	REQUEST =LOCK	UNCONDITIONAL
BART	'BLANK'	C48CE96572CA	N/P			OB =GLWSEMP	STATE =X	ZPARM INTERVAL= 60
DSNREXX	DB2CALL					PAGE=X'00010B'	DURATION=COMMIT	INTERV.COUNTER= 1
							HASH =X'004B490B'	
							----- HOLDERS/WAITERS -----	
							HOLDER	
							LUW=USIBMSC.SCPDB9A.C48CE974E055	
							MEMBER =N/P	
							CONNECT =DB2CALL	
							PLANNAME=DSNREXX	
							CORRID =BARTSQR4	
							DURATION=COMMIT	
							PRIMAUTH=BART	
							STATE =X	

### IFCID 313: long running units of recovery

Besides monitoring for deadlocks and timeouts (these are cases where a fault has already occurred, as DB2 had to take action and preempt one of the threads in the system), IFCID 313 also acts as an “early warning” system. It alerts you to situations where things are still okay at this time, but can easily lead to a real problem situation resulting in a timeout or deadlock.

IFCID 313 records are created when DB2 statistics trace class 3 is active.

There are three variations of IFCID 313 records, all governed by a DSNZPARM:

- ▶ When a UR has written more than URLGWTH (DSNZPARM) log records without committing.
- ▶ When a UR had been active for more than URCHKTH (DSNZPARM) system checkpoints without committing.
- ▶ When thread has been holding a read claim for more than LRDRTHLD (DSNZPARM) minutes without committing.

**Tip:** The default value for URLGWTH, URCHKTH, and LRDRTHLD is zero, which means these features are disabled. Make sure to activate these DSNZPARMs, as they can alert you to potential concurrency issues.

IFCID 313 records can be processed by OMEGAMON PE in batch (with PK93904 and PTF UK49345) or you can use IFI reads to trap them and have a program send alerts to people based on this information (which is what OMEGAMON PE exception event processing is doing. Refer to 8.3, “Periodic monitoring for concurrency problems” on page 227 for more information).

### **Exceeding URLGWTH number of log records without a commit**

When a unit of recovery writes more than URLGWTH log records without committing, a DSNJ031I (refer to “DSNJ031I: uncommitted UR with many log records” on page 225) is written to the console and an IFCID 313 is produced.

There are no specific commands to process IFCID 313 records, so the easiest way to analyze them in batch is by way of the RECTRACE command:

```
RECTRACE
TRACE
INCLUDE (IFCID(313))
LEVEL(LONG)
```

The RECTRACE report is shown in the following examples of the three different types of IFCID 313 records.

Example 8-28 shows the output of a RECTRACE report.

*Example 8-28 More than URLGWTH log records*

PRIMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME				TRANSACTION
ORIGAUTH	CORRNAME	CONNTYPE	RECORD TIME	DESTNO ACE	IFC	DESCRIPTION	DATA	
PLANNAME	CORRNMBR		TCB CPU TIME	ID	ID			
N/P	N/P	C48E0B328F57	N/P	N/P			N/P	
N/P	N/P	'BLANK'	04:14:03.09811284	11071	1 313	UNCOMMITTED	NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1	
N/P	N/P		N/P			UNIT OF RECOV		
-----								
UNCOMMITTED URID : X'00184F220C42'				CHKPTS TAKEN :	1	TYPE OF UR/UW: FL		
NETWORKID : USIBMSC				LUNAME :	SCPDB9A	INSTANCE :	C48E0B328F57 COMMIT COUNT : 1	
CONNECTION ID : TSO				CORRELATION ID:	BART	MESSAGE NUMBER: DSNJ031I		
PLAN NAME : DSNESPCS				WS NAME :	N/P	TRANSACTION: N/P		
TYPE OF THRESHOLD: LOG RECORDS				# LOG RECORDS WRITTEN:	100000			
AUTHORIZATION ID :BART								
END USER USERID :N/P								
-----								

In this case, the threshold type is the number of log records, and the DSNZPARM threshold value is 100,000. The trace record also shows the corresponding console message that will be produced, namely DSNK031I).

The trace record contains all the necessary information for easy identification of the offending unit of recovery, such as:

- ▶ The URID (in case you need to print the DB2 recovery log)
- ▶ The LUW-ID
- ▶ The connection-ID
- ▶ The correlation-ID
- ▶ The plan name
- ▶ The workstation name
- ▶ The transaction name

- The authorization ID
- The user ID

**Note:** Make sure to have the PTF for APAR PK93904 (PTF UK49345) applied in order to correctly format IFCID 313.

As there is normally a strong relationship between the number of log records that are written and the number of locks that are acquired, there is a good chance that this “problem” transaction will also cause concurrency problems and will need to be investigated.

### ***Exceeding URCHKTH number of system checkpoints without a commit***

When a unit of recovery is active (in-flight) for more than URCHKTH system checkpoints without committing, a DSNR035I (refer to “DSNR035I: uncommitted UR for a number of system checkpoints” on page 226) is written to the console and an IFCID 313 is produced.

Example 8-29 shows the output of a RECTRACE report.

*Example 8-29 More than URCHKTH system checkpoints*

PRIMAUTH ORIGAUTH PLANNAME	CONNECT CORRNAME CORRNMBR	INSTANCE CONNTYPE	END_USER RECORD TIME TCB CPU TIME	WS_NAME DESTNO ACE IFC ID	DESCRIPTION	TRANSACTION DATA
N/P	N/P	C48C891C4A79	N/P	N/P		N/P
N/P	N/P	'BLANK'	04:16:24.98762559	11092	2 313 UNCOMMITTED	NETWORKID: DB9A
N/P	N/P	N/P	N/P		UNIT OF RECOV	LUNAME: SCPDB9A LUWSEQ: 1
-----						
UNCOMMITTED URID : X'00185238E0D2'						
CHKPTS TAKEN : 3						
TYPE OF UR/UW: FL						
LUNAME : USIBMSC						
INSTANCE : C48E0B680FF6						
COMMIT COUNT : 1						
CONNECTION ID : TSO						
CORRELATION ID: BART						
MESSAGE NUMBER: DSNR035I						
PLAN NAME : DSNESPCS						
WS NAME : N/P						
TRANSACTION: N/P						
TYPE OF THRESHOLD: CHECKPOINTS						
# LOG RECORDS WRITTEN: 447						
AUTHORIZATION ID :BART						
END USER USERID :N/P						
-----						

In this case, the threshold type is the number of system checkpoints, and the DSNZPARM threshold value is 3 in our system. The trace record also shows the corresponding console message that will be produced, namely DSNR035I).

The trace record contains all the necessary information to easily identify the offending unit of recovery, such as:

- The URID (in case you need to print the DB2 recovery log)
- The LUW-ID
- The connection-ID
- The correlation-ID
- The plan name
- The workstation name
- The transaction name
- The authorization ID
- The user ID

**Note:** Make sure that you have the PTF for APAR PK93904 applied in order to correctly format IFCID 313.

In case there is an in-doubt UR that has been outstanding for more than URCHKTH system checkpoints, a ‘DU’ type (in-doubt) IFCID 313 trace record is produced and a DSNR036I message is written to the console.

### ***Holding a read claim for more than LRDRTHL D minutes without a commit***

This variation of the IFCID 313 trace record is slightly different from the previous ones, as it is not related to an outstanding unit of recovery. It applies to threads that are holding a read claim for longer than LRDRTHL D (DSNZPARM) minutes. It applies only to reader transactions, that is, transactions that do not have an open unit of recovery (open URs are governed by the other two DSNZPARMS described above).

In this case, the trace record is produced because a long running reader has been holding a read claim for more than three minutes, that is, the LRDRTHL D value on our system.

Example 8-30 shows the formatted output of a long reader warning IFCID 313 trace record. Note that the threshold type is READER in this case.

**Example 8-30** *Holding a read claim for more than LRDRTHL D minutes*

PRIMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME	RECORD TIME	DESTNO	ACE	IFC	DESCRIPTION	TRANSACTION
ORIGAUTH	CORRNAME	CONNTYPE	TCB CPU TIME	DESTNO	ACE	IFC	DESCRIPTION	TRANSACTION	DATA	
PLANNAME	CORRNMBR									
N/P	N/P	C48C89207193	N/P	N/P						
N/P	N/P	'BLANK'	04:18:11.58485370	11111	3	313	UNCOMMITTED	NETWORKID: DB9A	LUNAME: SCPDB9A	LUWSEQ: 1
N/P	N/P		N/P				UNIT OF RECOV			
-----										
UNCOMMITTED URID : 'BLANK' CHKPTS TAKEN : 0 LUW ID : USIBMSC SCPDB9A D + TYPE OF UR: RR										
CONNECTION ID : TSO CORRELATION ID: BART MESSAGE NUMBER: 'BLANK'										
PLAN NAME : DSNESPCS WS NAME : N/P TRANSACTION: N/P										
TYPE OF THRESHOLD: READER TOTAL NUMBER OF MINUTES: 3										
AUTHORIZATION ID :BART										
END USER USERID :N/P										
-----										

As for the other IFCID 313 types, this trace record also contains all the necessary information to easily identify the offending thread, such as:

- ▶ The LUW-ID
- ▶ The connection-ID
- ▶ The correlation-ID
- ▶ The plan name
- ▶ The workstation name
- ▶ The transaction name
- ▶ The authorization ID
- ▶ The user ID

Note that there is no console message associated with this type of IFCID 313 record.

It is important not only to monitor for long running units of recovery (which is done by way of the DSNZPARMS that we discussed in the two previous sections), but also for long running readers (holding a read claim on an object for a long time). A long claim can prevent utilities from breaking in, as utilities tend to drain an object, but have to wait for all the claimers to release their claim before they can take over the object.

### **IFCID 337: lock escalation**

Another interesting IFCID that is easy and cheap to track, yet can provide information about potential concurrency problems, is IFCID 337. It is also created when DB2 statistics trace class 3 is active.

IFCID 337 complements the DB2 accounting information that we analyze in 8.3.3, “Analyzing DB2 accounting data” on page 236. When the DB2 accounting records indicate that lock escalation has occurred, you can use IFCID 337 to identify on which object(s) the escalation occurred, and the statement that was being executed at the time the escalation occurred.

Example 8-31 shows a formatted IFCID 337 record using the OMEGAMON PE RECTRACE command. The record contains all the necessary information to identify the thread that incurred the lock escalation. You can use the LUW-ID (network-id.luname.instance-number.commit-sequence-number) to match the accounting record with the IFCID 337 record.

*Example 8-31 IFCID 337: lock escalation*

LOCATION: DB9A		OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)					PAGE: 1-1	
GROUP: N/P		RECORD TRACE - LONG					REQUESTED FROM: NOT SPECIFIED	
MEMBER: N/P							TO: NOT SPECIFIED	
SUBSYSTEM: DB9A							ACTUAL FROM: 07/29/09 04:13:58.55	
DB2 VERSION: V9							PAGE DATE: 07/29/09	
PRIMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME	TRANSACT			
ORIGAUTH	CORRNAME	CONNTYPE	RECORD TIME	DESTNO ACE	IFC	DESCRIPTION	DATA	
PLANNAME	CORRNMBR		TCB CPU TIME		ID			
-----								
BART	TSO	C48E0B328F57	'BLANK'	'BLANK'			'BLANK'	
BART	BART	TSO	04:13:58.55988928	11070	1 337	LOCK ESCAL.	NETWORKID: USIBMSC	LUNAME: SCPDB9A LUWSEQ: 1
DSNESPSCS	'BLANK'		N/P			OCCURENCES		
-----								
DATABASE ID		: GLWBART		PAGESET/TABLE ID: GLWSEMP		STATEMENT NUMBER :		251
LOCK STATE		: N/P		LOWER LOCK TYPE : PAGE LOCK		NUMBER LOWER LOCKS:		501
WAITERS STATMT ID:		499						
COLLECTION ID		: DSNESPSCS						
PACKAGE NAME		: DSNESM68						
-----								

In this case, the lock escalation occurred on the GLWBART.GLWSEMP table space. 501 page or row locks were replaced by a single partition lock (as GLWSEMP is a partitioned table space). The LOCKMAX value for GLWSEMP is 500. Note that the lock state value is N/P in the output. This field normally contains the lock state that was escalated to, but the field is not populated when selective partition locking is being used, which is always used, because V8 is for partitioned table spaces.

IFCID 337 also contains the statement number that was executing at the time the lock escalation occurred. The statement number is 251 in Example 8-31. Figure 8-2 shows some of the statements from the SPUFI package that was used to executed the statement that escalated.

As you can see, statement 251 is just the DECLARE CURSOR statement for a dynamic SQL statement. It is not very helpful to determine which is the actual SQL statement that got escalated. If the escalating statement had been a static SQL statement, looking at the package information would have allowed you to determine the statement that was escalated.

```

DB2 Admin ----- Extracted SQL ----- Columns 00001 00072
Command ==>                                     Scroll ==> CSR

***** Top of Data *****
=NOTE= -- SQL statements in PACKAGE : DSNESPSCS.DSNESM68.( )
=NOTE= -- SQL in stmt: 251
000001 DECLARE C1 CURSOR FOR S1
=NOTE= -- SQL in stmt: 324
000002 PREPARE S1 INTO :H FROM :H

```

*Figure 8-2 Excerpt from DSNESM68 SQL statements*

When the statement is in the dynamic statement cache, the 'WAITERS STATMT ID' will contain the cached statement ID. In our example, the statement ID is 499.

Figure 8-3 shows an excerpt of the information that is in the DSN\_STATEMENT\_CACHE\_TABLE after running an EXPLAIN STMTCACHE ALL statement. STMT\_ID 499 is a statement executed by SPUFI.

STMT_ID	PROGRAM	INV_DROPALT	INV_REVOKE	INV_LRU	INV_RUNSTA	CACHED_TS	USERS	COPIES	PRIMAUTH	CURSQ	BIND_QUALI	BIND_JSO	BIND_CDATA
441	DSNREXX	N	N	N	N	2009-07-28 22:43...	0	0	DB2R1	DB2R1	DB2R1	CS	N
254	DSNREXX	N	N	N	N	2009-07-28 19:35...	0	0	DB2R3	DB2R3	DB2R3	CS	Y
45	FPE@WRPA	N	N	N	N	2009-07-27 19:28...	0	0	STC	STC	STC	UR	N
17	DSNREXX	N	N	N	N	2009-07-27 19:27...	0	0	DB2R1	DB2R1	DB2R1	CS	N
67	DSNREXX	N	N	N	N	2009-07-28 02:32...	0	0	BART	BART	BART	CS	N
139	DSNREXX	N	N	N	N	2009-07-28 14:21...	0	0	BART	BART	BART	CS	Y
491	DSNREXX	N	N	N	N	2009-07-28 23:16...	0	0	DB2R1	DB2R1	DB2R1	CS	N
499	DSNREXX	N	N	N	N	2009-07-28 23:49...	0	0	BART	BART	BART	CS	Y
97	SYSLH200	N	N	N	N	2009-07-28 13:27...	0	0	DB2R3	DB2R3	DB2R3	CS	N
88	DSNREXX	N	N	N	N	2009-07-28 13:25...	0	0	DB2R1	DB2R1	DB2R1	CS	N
34	FPE@WRPA	N	N	N	N	2009-07-27 19:28...	0	0	STC	STC	STC	UR	N
426	ADBMAIN	N	N	N	N	2009-07-28 22:34...	0	0	DB2R1	DB2R1	DB2R1	CS	N
56	DSNREXX	N	N	N	N	2009-07-28 02:32...	0	0	BART	BART	BART	CS	N
429	DSNREXX	N	N	N	N	2009-07-28 22:43...	0	0	DB2R1	DB2R1	DB2R1	CS	N
102	DSNREXX	N	N	N	N	2009-07-28 14:04...	0	0	DB2R1	DB2R1	DB2R1	CS	N
74	DSNREXX	N	N	N	N	2009-07-28 02:32...	0	0	BART	BART	BART	CS	N
62	DSNREXX	N	N	N	N	2009-07-28 02:32...	0	0	BART	BART	BART	CS	N

Figure 8-3 List dynamic statement cache

Figure 8-4 shows the SQL statement text that is associated with STMT\_ID 499. It is an INSERT FROM SELECT statement. As the GLWTEMP table contains quite a large number of rows, it is not unusual for lock escalation to occur.

```

INSERT INTO GLWBART.GLWTEMP SELECT EMP_NO +
50000 ,FIRSTNME ,MIDINIT ,LASTNAME ,WORKDEPT
,PHONENO ,HIREDATE ,JOB ,MANAGER ,EDLEVEL ,SEX
,BIRTHDATE ,SALARY ,BONUS ,COMM ,CREATED_TS
,CREATED_BY ,UPDATED_TS ,UPDATED_BY FROM
GLWBART.GLWTEMP

```

Figure 8-4 SQL statement text

Note that this is not necessarily the statement that is really responsible for acquiring most of the locks, although it most certainly is in this case. It may be that the statement that triggered the lock escalation acquired only a single lock, but it happened to be the one that made DB2 exceed the table space's / partition's LOCKMAX value (or NULMKTS if LOCKMAX SYSTEM is used) and triggered escalation. But in any case, it is worthwhile to discover which statement triggered the lock escalation.

### 8.3.5 Exception monitoring using OMEGAMON PE

There are many ways to do exception monitoring, whether it is done online or in batch (after the fact). Discussing all of them is beyond the scope of this publication. We only provide a few options you can exploit using OMEGAMON PE. The important point is that you need to set up a strategy in your installation to check for exception conditions on a regular basis. These exceptions are of course not limited to conditions that are related to concurrency issues.

A way to track the health of your DB2 system or an application is by setting up exception processing rules. If something out of the ordinary happens, an exception gets triggered and informs you about this anomaly.

OMEGAMON PE allows for *event exception* processing and *periodic exception* processing:

- ▶ Event exception processing allows you to monitor a DB2 subsystem for the occurrence of a number of particular events. For example, you can set up OMEGAMON PE to monitor events like timeouts, deadlocks, or long running URs. If one of those occurs, an exception is triggered.
- ▶ Periodic exception processing allows you to periodically monitor thread activity and statistics fields for exception conditions.

OMEGAMON PE can give an exception alert back to the user, or send the alert to a user exit. You can use the Exception Processing window of the OMEGAMON PE workstation client to start and stop periodic exceptions or event exceptions, as well as to view the exception processing results. You can also start and stop exception monitoring from the OMEGAMON PE ISPF interface. When a periodic exception or an event exception occurs, you are immediately alerted if you are logged on to the workstation or ISPF online monitor. However, if you are not be logged on when the exception occurs, the exception is still logged, so you can review it afterwards.

Another way to do exception monitoring is by using *exception triggering with screen logging* by way of the classic interface. In general, exception analysis with the classic interface is of limited usefulness in tracking exceptions, because the exception is only visible during the cycle (a typical cycle is five seconds) in which it is triggered. Unless an exception is triggered over many consecutive cycles, there may not be enough time to investigate the thread causing the exception before the condition disappears. To get around this problem, you can use exception triggering combined with automatic screen logging, which is a special feature of the OMEGAMON PE classic interface. When an exception has been set up with automatic screen logging, each time the exception occurs, a specified panel is automatically logged for the thread that caused the exception.

*Exception reporting* is yet another option available to you. OMEGAMON PE allows you to generate accounting and statistics exception reports as well as specific exception reports to help you identify performance problems in your DB2 subsystem. You first set up a set of exceptions and their threshold values using the ISPF interface, and then use the EXCEPTION keyword on the ACCOUNTING or STATISTICS TRACE, REPORT or FILE command while also providing the exception threshold data set. The produced reports/files contain values outside user-specified limits.

For more details, refer to *A Deep Blue View of DB2 Performance: IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SG24-722424 and *IBM DB2 Performance Expert for z/OS Version 2*, SG24-68677.







## Analyzing concurrency problems

In this chapter, we discuss some methods for analyzing locking problems.

The purpose of this chapter is to explain the tools you have at your disposal to address concurrency problems and provide you with a set of strategies for solving various types of locking, serialization, and concurrency issues.

The following topics are covered:

- ▶ In 9.1, “The analysis toolbox” on page 256, we discuss the different tools that are available for providing information about locking, such as DB2 commands and DB2 traces.
- ▶ In 9.2, “Analysis of a simple deadlock scenario” on page 280, we go through a deadlock problem and show how to identify the root cause of the problem.

## 9.1 The analysis toolbox

In this section, we describe the different tools you have at your disposal to investigate a concurrency problem in order to discover the root cause of the problem. The toolbox consists of:

- ▶ DB2 commands
- ▶ EXPLAIN statement and the information in the dynamic statement cache
- ▶ Standard DB2 traces
- ▶ Detailed DB2 traces
- ▶ Using online monitor

### 9.1.1 DB2 commands

We describe two types of commands:

- ▶ DISPLAY DATABASE commands
- ▶ DISPLAY THREAD commands

#### DISPLAY DATABASE commands

The -DISPLAY DATABASE command is a very useful command for obtaining information about the status and usage of the objects (table spaces, partitions, indexes, and index partitions) in that database. The -DISPLAY DATABASE command has a number of options that allow you to display different types of information about the database. The keywords that are most often used when dealing with concurrency issues are:

<b>USE</b>	This option allows you to do a quick check to see if a transaction, job, and so on is accessing (holding locks or waiting for them) the displayed object. It displays information, such as the connection-IDs, correlation-IDs, and authorization IDs, and the LUW-ID and location of any remote threads accessing the local database.
<b>RESTRICT</b>	This option lists the objects that have a restrictive state. There are many types of restrictive states (refer to 3.7, “Restricted states” on page 56) that can prevent an application from accessing an object. When the system is not performing well and is generating many DSNT500I (resource unavailable) messages, it is always good to make sure that no objects are in a restricted state that can prevent transactions, batch jobs, or utilities from accessing the table or index spaces.
<b>CLAIMERS</b>	This option lists the claims on all table spaces, index spaces, and (physical and logical) partitions whose statuses are displayed, as well as information that allows you to identify who acquired the claim, such as the LUW-ID and location of any remote threads accessing the local database, and the connection-IDs, correlation-IDs, and authorization IDs, as well as the agent token number for the claim, if the claim is local. You can then match the token with the output of the -DIS THREAD command to discover more information about the thread.

Example 9-1 shows the output of a -DIS DB CLAIMERS command. You can see that the TSO user BART (token 10737) had a write claim (WR) and a cursor stability read claim (CS) on table space TSBART2. Both will be released at commit (C) time, which is normally the case for claims.

Example 9-1 DISPLAY DATABASE CLAIMERS

```

DSNT360I  -DB9A *****
DSNT361I  -DB9A *   DISPLAY DATABASE SUMMARY
              *   GLOBAL CLAIMERS
DSNT360I  -DB9A *****
DSNT362I  -DB9A      DATABASE = DBBART2  STATUS = RW
              DBD LENGTH = 4028
DSNT397I  -DB9A
NAME      TYPE PART  STATUS              CONNID   CORRID      CLAIMINFO
-----
TSBART2   TS        RW                TSO      BART        (WR,C)
-          AGENT TOKEN 10737
TSBART2   TS        RW                TSO      BART        (CS,C)
-          AGENT TOKEN 10737
***** DISPLAY OF DATABASE DBBART2  ENDED *****
DSN9022I  -DB9A DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***

```

## LOCKS

Specifying the LOCKS option provides you with information about:

- The transaction locks (L-locks) for all table spaces, tables, index spaces, and partitions (physical and logical) whose statuses are displayed.
- Identification information, such as the correlation-IDs, LUW-ID, and authorization-IDs for all applications allocated to these objects.
- The drain locks for a resource held by running jobs.
- The retained locks for a resource.
- The page set or partition physical locks (P-locks) for a resource.
- The agent token for the lock holder, if the lock holder is local (you can match it with the output of the -DIS THREAD command).

Note that the LOCKS option provides no information about the number of lower level lock (page, row, or LOB or XML) that are held on an object. It only provides information about locks at the table space, partition, index (logical and physical), and table level.

Example 9-2 shows a sample output of a -DIS DB LOCKS command.

*Example 9-2 DISPLAY DATABASE LOCKS*

```

DSNT360I  -DB9A *****
DSNT361I  -DB9A *   DISPLAY DATABASE SUMMARY
              *   GLOBAL LOCKS
DSNT360I  -DB9A *****
DSNT362I  -DB9A      DATABASE = DBBART2  STATUS = RW
              DBD LENGTH = 4028
DSNT397I  -DB9A
NAME      TYPE PART  STATUS              CONNID   CORRID   LOCKINFO
-----
TSBART2   TS        RW                TSO      BART     H-IX,S,C
-          -          AGENT TOKEN 10737
3         TB        TSO      BART     H-X,T,C
-          -          AGENT TOKEN 10737
***** DISPLAY OF DATABASE DBBART2  ENDED *****

```

As the LOCKINFO field in Example 9-2 is not always easy to read, we use the information in Table 9-1 to explain how to interpret the -DIS DB LOCKS output. The format of the LOCKINFO field is always in the following format:

LOCK-STATUS - LOCK-STATE , LOCK-TYPE , LOCK-DURATION

For Example:

H - IX , S , C

In Example 9-2, the LOCKINFO field shows that the agent with TOKEN 10737 is holding (H) an IX, table space logical lock (S), and it is held until the transaction commit (lock duration is C).

*Table 9-1 LOCKINFO information*

Status	State		Type	Duration
	L-lock + drain lock	P-lock		
H: Holding	IS	IX	S: TS L-lock	H: A: Allocation C: Commit H: Close cursor M: Freed by system P: Plan complete I: Interest (PS P-lock)
W; Waiting	IX	IX	T: TB L-lock	
R: Retained	S	S	P: PT L-lock	
	U <sup>ab</sup>		C: CS drain lock	
	SIX <sup>b</sup>	SIX	R; RR drain lock	
	X	X	W: WR drain lock	W: Number in the waiting queue
		NSU	PP: PS P-lock	R; N/A

a. Not applicable to LOBs

b. Not applicable to drain locks

The abbreviations in Table 9-1 on page 258 are defined as follows:

<b>TS LL</b>	Table space <i>logical lock</i>
<b>TB LL</b>	Table logical lock
<b>PT LL</b>	Partition logical lock
<b>CS DL</b>	Cursor stability <i>drain lock</i>
<b>RR DL</b>	Repeatable read drain lock
<b>WR DL</b>	Write drain lock
<b>PS PL</b>	Page set or partition <i>P-lock</i>

The major drawback of using the -DIS DB command is that it only shows the information at the point in time that the command is issued. It can be that by the time you issue the command, the problem is already resolved and that the commands will not be able to show anything.

## DISPLAY THREAD commands

The -DISPLAY THREAD command can be helpful in identifying users that are active in the system. The command comes in many flavors, but in this section we only focus on a few of them. The -DISPLAY DATABASE command that we discussed in “DISPLAY DATABASE commands” on page 256 provides some information about a thread, including the thread token. You can use the thread token from the -DISPLAY DATABASE command and match it with the thread token from the -DISPLAY THREAD output (see Example 9-3).

Token 10737 is matched with the token in the output of the -DISPLAY DATABASE LOCKS command in Example 9-2 on page 258. You can see that the user is a TSO user (allied thread) that is using SPUFI (plan name is DSNESPCS) and that there is a thread allocated, but that thread is currently not active in DB2 (no “\*” in the ‘A’ column).

Example 9-3 -DIS THD(\*) output

---

```

DSNV401I  -DB9A DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -DB9A ACTIVE THREADS -
NAME      ST A   REQ ID          AUTHID   PLAN      ASID  TOKEN
RRSAF     T    18546             STC      K02PLAN   00A2  7544
RRSAF     T    7425              STC      K02PLAN   00A2  7545
RRSAF     T    9340              STC      K02PLAN   00A2  7546
RRSAF     T   20348             STC      DB2PM     00A2  7547
RRSAF     T   14380             STC      K02PLAN   00A2  7548
RRSAF     T    7676 OMEGAMON     STC      K02PLAN   00A2  7549
RRSAF     T      7 DB9AADMT0066 STC      ?RRSAF   00A5    2
RRSAF     T   28695 DB9AADMT0001 STC      ?RRSAF   00A5    3
TSO      T      30 BART        BART      DSNESPCS 00BB 10737
TSO       T   *      3 BART        BART      00BB 10751
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I  -DB9A DSNVDT '-DIS THD' NORMAL COMPLETION

```

---

For distributed threads, you also have the information about the workstation name, the client user ID, and the application name. This is shown in Example 9-4 for an active thread.

*Example 9-4 -DIS TH(\*) output*

---

```

DSNV401I  -D9C1 DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -D9C1 ACTIVE THREADS -
NAME      ST A   REQ ID          AUTHID   PLAN      ASID TOKEN
D9C1      RA *    0 028.DBAA 02 SYSOPR          00F9  121
V445-G90C0595.P879.C4A044F395FC=121 ACCESSING DATA FOR
::9.12.5.149
SERVER   RA *    245 db2jcc_appli PAOLOR7  DISTSERV 00F9  118
V437-WORKSTATION=kodiak.itso.ibm., USERID=PAOLOR7,
APPLICATION NAME=db2jcc_application
V445-G90C0595.P876.C4A044F2EEA4=118 ACCESSING DATA FOR
::9.12.5.149
SERVER    RA *    102 db2jcc_appli PAOLOR7  DISTSERV 00F9  114
V437-WORKSTATION=kodiak.itso.ibm., USERID=PAOLOR7,
APPLICATION NAME=db2jcc_application
V445-G90C0595.P86E.C4A044F13E06=114 ACCESSING DATA FOR
::9.12.5.149
SERVER    RA *    203 db2jcc_appli PAOLOR7  DISTSERV 00F9  109
V437-WORKSTATION=kodiak.itso.ibm., USERID=PAOLOR7,
APPLICATION NAME=db2jcc_application
V445-G90C0595.P868.C4A044F05F85=109 ACCESSING DATA FOR
::9.12.5.149
SERVER    RA *    207 db2jcc_appli PAOLOR7  DISTSERV 00F9  116
V437-WORKSTATION=kodiak.itso.ibm., USERID=PAOLOR7,
APPLICATION NAME=db2jcc_application
V445-G90C0595.P874.C4A044F2D79B=116 ACCESSING DATA FOR
::9.12.5.149
TSO       T *     3 BART          BART          00E2  128
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I  -D9C1 DSNVDT '-DIS THD' NORMAL COMPLETION

```

---

Example 9-5 shows the output for an inactive DBAT.

*Example 9-5 -DIS THD(\*) TYPE(INACTIVE) output*

---

```

DSNV401I  -DB9A DISPLAY THREAD REPORT FOLLOWS -
DSNV424I  -DB9A INACTIVE THREADS -
NAME      ST A   REQ ID          AUTHID   PLAN      ASID TOKEN
SERVER   R2      0 db2bp.exe    BART     DISTSERV 00AA 15083
V437-WORKSTATION=L3E2486, USERID=bart,
APPLICATION NAME=db2bp.exe
V445-G91E1CCA.J505.037A01222834=15083 ACCESSING DATA FOR
::9.30.28.202
DISPLAY INACTIVE REPORT COMPLETE

```

---

### ***Displaying system threads***

Before DB2 9, you could only display user threads using the -DIS THD command. Version 9 introduced a new option called TYPE(SYSTEM). It allows you to list (some of) the DB2 system threads. These are threads that are used by DB2 to process work requests on behalf of user threads, such as prefetch requests, or run system related activities, such as pseudo-close processing or system checkpoint activity.

Sometimes a resource unavailable message, a deadlock, or timeout message contains a system thread identifier. An example is shown in Example 9-6. In this case, the system thread is 010.PM2PCP01. To learn more about what type of work this thread represents, refer to Chapter 34, “System-Agent Correlation Identifiers”, in *DB2 Version 9.1 for z/OS Diagnosis Guide and Reference*, LY37-3218. In this case, 010.PM2PCP01 is the pseudo-close processor service task.

*Example 9-6 Timeout with a system thread involved*

---

```
DSNT376I -DB9A PLAN=DISTSERV WITH 493
CORRELATION-ID=db2jcc_appli
CONNECTION-ID=SERVER
LUW-ID=RFF5B99F.X65E.C462790600EF=81684
THREAD-INFO=ZWAEX01:za2alumni00171:zwaewx01:db2jcc_applicatio
IS TIMED OUT. ONE HOLDER OF THE RESOURCE IS PLAN=
WITH
CORRELATION-ID=010.PM2PCP01
CONNECTION-ID=DB9A
LUW-ID=USIBMSC.SCPDB9A.C45E76E03C5F
THREAD-INFO=SYSOPR:*.*:.*
ON MEMBER DB9A
```

---

If this timeout condition persists for a long period of time, running -DIS THD(\*) TYPE(SYSTEM) can sometimes help identify what this system thread is doing. An excerpt of the command output is provided in Example 9-7.

*Example 9-7 -DIS THD(\*) TYPE(SYSTEM) output*

---

```
DSNV401I -DB9A DISPLAY THREAD REPORT FOLLOWS -
DSNV497I -DB9A SYSTEM THREADS -
DB2 ACTIVE
```

NAME	ST	A	REQ	ID	AUTHID	PLAN	ASID	TOKEN
DB9A	N	*	0	002.VMON	03 SYSOPR		00AA	0
V507-INACT MONITOR, INTERVALS=7219, STG=N/A, BOOSTS=N/A, HEALTH=N/A								
DB9A	N	*	0	028.ERRMON01	SYSOPR		00AA	0
DB9A	N	*	0	028.RESYNT01	SYSOPR		00AA	0
DB9A	N	*	0	027.OPNACB01	SYSOPR		00AA	0
V490-SUSPENDED				09208-19:26:18.70	DSNLQCTL	+00000A92	UK40815	
DB9A	N	*	0	027.DDFINT03	SYSOPR		00AA	0
V490-SUSPENDED				09208-19:26:18.70	DSNLQCTL	+00000A92	UK40815	
DB9A	N	*	0	027.DDFINT04	SYSOPR		00AA	0
V490-SUSPENDED				09208-19:26:18.70	DSNLQCTL	+00000A92	UK40815	
DB9A	N	*	0	027.SLSTNR02	SYSOPR		00AA	0
DB9A	N	*	0	027.RLSTNR02	SYSOPR		00AA	0
DB9A	N	*	0	027.GQRQST02	SYSOPR		00AA	0
V490-SUSPENDED				09223-20:06:38.18	DSNLQCTL	+00000A92	UK40815	
DB9A	N	*	0	028.RESYNC01	SYSOPR		00AA	0
V490-SUSPENDED				09208-19:26:18.91	DSNLQCTL	+00000A92	UK40815	
DB9A	N	*	0	010.PMICMS01	SYSOPR		00A9	0
V490-SUSPENDED				09223-20:18:06.98	DSNB1CMS	+000005C0	UK46416	
DB9A	N	*	0	010.PMITMR02	SYSOPR		00A9	0
DB9A	N	*	0	022.SPQMON01	SYSOPR		00A9	0
DB9A	N	*	0	014.RTSTST00	SYSOPR		00A9	0
V490-SUSPENDED				09223-20:22:24.49	DSNB1TMR	+00000B78	09.24	
DB9A	N	*	0	<b>010.PM2PCP01</b>	SYSOPR		00A9	0
<b>V490-SUSPENDED</b>				<b>09223-20:22:06.99</b>	<b>DSNB1TMR</b>	<b>+00000B78</b>	<b>09.24</b>	

```

DB9A      N  *      0 010.PM2PCK02 SYSOPR          00A9      0
V490-SUSPENDED 09223-20:22:29.14 DSNB1TMR +00000B78 09.24
DB9A      N  *      0 002.VMON 02 SYSOPR          00A9      0
V507-INACT MONITOR, INTERVALS=10829, STG=N/A, BOOSTS=N/A, HEALTH=N/A
DB9A      N  *      0 023.GCSCNM03 SYSOPR          00A8      0
DB9A      N  *      0 004.JTIMR 00 SYSOPR          00A8      0
DB9A      N  *      0 004.JW007 01 SYSOPR          00A8      0
V490-SUSPENDED 09223-20:22:50.15 DSNJW107 +0000029E UK22636
...

```

---

In this case, the service task is suspended in DSNB1TMR, which is a timer pop wait, so the pseudo-close processor is no longer doing any work, and is waiting again for the timer to expire. In this case, the -DISPLAY command was issued too late after the fact.

## 9.1.2 EXPLAIN

You can use the EXPLAIN SQL statement, or the EXPLAIN(YES) BIND option, to obtain information about the access path that an SQL statement (or the SQL statements in a plan or package) will use.

You can also use the STMTCACHE option of the EXPLAIN statement to extract information about SQL statements in the (global) dynamic statement cache (DSC).

### TSLOCKMOD column in the PLAN\_TABLE

The PLAN\_TABLE that is populated by executing the EXPLAIN statement not only contains information about the access path of the SQL statements, but also provides an indication of the mode of lock that will be acquired on either the new table, its table space, or table space partition. The values depend on whether or not the isolation can be determined at bind, prepare, or explain time.

This column only applies to the table space / table lock mode. It does not indicate whether locks will be acquired at the row or page level, or whether those will be S, U, or X locks.

Refer to 5.12, “Access path influence on locking” on page 138 for more information.

### EXPLAIN STMTCACHE

You can use the EXPLAIN STMTCACHE statement in several ways.

#### *Finding the SQL statement text*

You can use the EXPLAIN STMTCACHE statement to extract information from the (global) dynamic statement cache and insert it into the DSN\_STATEMENT\_CACHE\_TABLE table. Information from the dynamic statement cache can be very helpful in identifying an SQL statement that is involved in a locking conflict. We already used this technique when we analyzed the lock escalation trace record; refer to “IFCID 337: lock escalation” on page 250 for more information about how to extract information from the dynamic statement cache.



### ***Finding the users of an object***

Before the dynamic statement cache was introduced, dynamic SQL statements would obtain a DBD-lock (a lock on the database descriptor) to prevent an ALTER or DROP of an object in the database while an SQL statement was being executed that used that object. Dynamic SQL statements that are stored in the dynamic statement cache no longer acquire a lock on the DBD. However, DB2 still needs a mechanism to track whether or not an object is currently being used by an SQL statement that resides in the statement cache.

In Example 9-8, we try to drop table TBBART3, but the statement fails because there are still users accessing TBBART3. Note that there is no lock request that times out, just a resource unavailable message in the joblog, and a DSNT500I message on the console.

---

#### ***Example 9-8 Resource unavailable at DROP TABLE time***

---

```
DROP TABLE TBBART3;
SQLERROR ON DROP      COMMAND, EXECUTE FUNCTION
RESULT OF SQL STATEMENT:
DSNT408I  SQLCODE = -904, ERROR:  UNSUCCESSFUL EXECUTION CAUSED BY AN UNAVAILABLE RESOURCE.
          REASON 00E70081,
          TYPE OF RESOURCE 00000A00, AND
          RESOURCE NAME BART.TBBART3

DSNT418I SQLSTATE  = 57011 SQLSTATE RETURN CODE
DSNT415I SQLERRP   = DSNXIDMH SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD   = 5 0 0 -1 0 0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD   = X'00000005' X'00000000' X'00000000' X'FFFFFFFF' X'00000000'
X'00000000' SQL DIAGNOSTIC INFORMATION
```

#### **00E70081**

Explanation: A DROP or ALTER statement was issued but the object cannot be dropped or altered. The object is referenced by a prepared dynamic SQL statement that is currently stored in the prepared statement cache and is in use by an application.

---

A DSNT500I message is also sent to the console and DB2 MSTR joblog, as shown in Example 9-9.

---

#### ***Example 9-9 DSNT500I console message with RC 00E70081***

---

```
23.39.46 STC08083  DSNT500I  -DB9A DSNXIDMH RESOURCE UNAVAILABLE  821
          821              REASON 00E70081
          821              TYPE 00000A00
          821              NAME  .BART.TBBART3
```

---

The information from the dynamic statement cache can be used to discover who is using the object that you are trying to drop, especially when the information is retrieved immediately after the -904 SQLCODE is received.

In order to discover these statements, you can extract all the information from the dynamic statement cache using EXPLAIN STMTCACHE ALL and search the DSN\_STATEMENT\_CACHE\_TABLE table.

Example 9-10 shows a sample SQL statement to look for an object in the statement cache table. This statement is used by an SPUFI user; the statement that he was executing was `SELECT * FROM TBBART3`.

*Example 9-10 Selecting users from an object in the DSC*

```

STMT_ID
, SUBSTR(PROGRAM_NAME,1,12) AS PROGRAM_NAME
, CACHED_TS
, USERS
, COPIES
, SUBSTR(PRIMAUTH,1,12) AS PRIMAUTH
, SUBSTR(CURSQLID,1,12) AS CURSQLID
, SUBSTR(BIND_QUALIFIER,1,12) AS BIND_QUALIFIER
, SUBSTR(STMT_TEXT,1,100) AS STMT_TEXT
FROM DSN_STATEMENT_CACHE_TABLE
WHERE STMT_TEXT LIKE '%TBBART3%'
AND (USERS > 0 OR COPIES > 0) ;
-----+-----+-----+-----+-----+-----+-----+...
      STMT_ID  PROGRAM_NAME  CACHED_TS                      USERS      COPIES  PRIMAUTH ...
-----+-----+-----+-----+-----+-----+-----+...
      1503    DSNESM68      2009-08-12-19.35.31.897479      1           1    BART      ...

...+-----+-----+-----+-----+-----+-----+-----+
...      CURSQLID      BIND_QUALIFIER  STMT_TEXT
...+-----+-----+-----+-----+-----+-----+-----+
...      BART          BART          SELECT * FROM TBBART3

```

Note that the information in the `DSN_STATEMENT_CACHE_TABLE` is snapshot information and is only valid at the time the snapshot is taken. When taken immediately after the `DROP` or `ALTER` request fails, for example, by automatically dumping the cache after the `DSNT500I` console message, there is a good chance to catch the statement that is currently using the object that is the subject of the `DROP` or `ALTER`.

The time that the transaction is waiting for the object to become available is reported as `OTHER SERVICE` wait time in the DB2 accounting record, as shown in Example 9-11.

*Example 9-11 CLASS 3 wait times for statements to be released*

CLASS 3 SUSPENSIONS	ELAPSED TIME	EVENTS
LOCK/LATCH(DB2+IRLM)	0.000000	0
SYNCHRON. I/O	0.000000	0
DATABASE I/O	0.000000	0
LOG WRITE I/O	0.000000	0
OTHER READ I/O	0.000000	0
OTHER WRTE I/O	0.000000	0
SER.TASK SWTCH	1:00.003078	13
UPDATE COMMIT	0.000000	0
OPEN/CLOSE	0.000000	0
SYSLGRNG REC	0.000000	0
EXT/DEL/DEF	0.000000	0
OTHER SERVICE	1:00.003078	13
ARC.LOG(QUIES)	0.000000	0
LOG READ	0.000000	0
DRAIN LOCK	0.000000	0
CLAIM RELEASE	0.000000	0
PAGE LATCH	0.000000	0

NOTIFY MSGS	0.000000	0
GLOBAL CONTENTION	0.000000	0
COMMIT PH1 WRITE I/O	0.000000	0
ASYNCH CF REQUESTS	0.000000	0
TCP/IP LOB	0.000000	0
TOTAL CLASS 3	1:00.003078	13

---

### 9.1.3 Standard DB2 traces

In 8.3.1, “Which information to gather” on page 227, we stated that you should permanently activate the following traces:

- Statistics trace class 1, 3, 4, and 5 with a STATIME of 1 minute
- Accounting trace class 1, (2), 3, 7, 8

Without information about how your system is performing, it is very likely that you will find yourself in unforeseen circumstances regarding the system.

### 9.1.4 Detailed DB2 traces

DB2 statistics and accounting information are very helpful in discovering whether there is a concurrency problem in the system and identifying the plan or package that is experiencing the concurrency problem, but this information is often not enough to identify why the problem occurs, and what or who is actually causing the problem.

To zoom in on a concurrency problem, it is often necessary to use more detailed DB2 traces. In this section, we give an overview of the types of information that are available to construct a complete picture of a concurrency problem.

The vast amount of information that the DB2 instrumentation facility can provide is a major advantage when using DB2 for z/OS. There are specific Instrumentation Facility Component IDentifiers (IFCIDs) trace records for almost any type of event that you would like to gather information about, and the number of IFCIDs that are related to locking and concurrency is rather large. To structure the information, we grouped them into a few categories related to way you normally analyze concurrency problems. These categories do not necessarily map to the DB2 trace classes.

#### Lock suspension information

When you have identified the plan or package that is experiencing a concurrency problem, it is not always obvious which objects are suspended. To identify the resources involved, review the following IFCIDs:

- 44 - 45: IRLM lock suspend and lock resume request
- 213 - 214: Start/end waiting for a drain lock
- 215 - 216: Start/end waiting for the claim count to go to zero
- 226 - 227: Start/end page latch wait
- 105, 107: Allow OBID translation from OBID numbers to object names

You can use the following START TRACE command to gather this information:

```
-START TRACE(P) CLASS(30) IFCID(44,45,105,107,213,214,215,216,226,227)
  TDATA(CPU COR DIST) DEST(SMF/GTF)
```

Note that this trace request does not gather information about DB2 latch contention. We discuss this topic in “DB2 latch suspension detailed information” on page 279.

This trace can generate a great deal of trace data and impacts the CPU processing for the transactions while the trace is active. We therefore recommend turning on the trace for a short period of time, and if possible, to limit the TRACE command using filters like AUTHID, PLANNAME, or some of the new filters that were introduced in DB2 9, such as PKGCOL, PKGPROG, APPNAME, WRKSTN, CONNID, and CORRID.

Once you gathered this trace information, you can use the OMEGAMON PE LOCK SUSPENSION report set to condense and summarize this information. A sample report excerpt is shown in Example 9-12. You can use the following command to generate this report:

```
DB2PM
  GLOBAL
    TIMEZONE(+2)
  LOCKING
  REPORT
    LEVEL(SUSPENSION)
    ORDER(DATABASE-PAGESET)
EXEC
```

The ORDER(DATABASE-PAGESET) keyword indicates that you want to do a rollup, or GROUP BY in SQL terminology, of the suspensions in the trace at the database-pageset (object) level. The report contains the details of each suspension that occurred, but it is also summarized at the database and pageset level.

**Example 9-12** Locking suspension report

LOCATION: WTSC03			OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)							PAGE: 1-2					
GROUP: N/P			LOCKING REPORT - SUSPENSION							REQUESTED FROM: NOT SPECIFIED					
MEMBER: N/P										TO: NOT SPECIFIED					
SUBSYSTEM: DB9A			ORDER: DATABASE-PAGESET							INTERVAL FROM: 06/25/09 07:22:42.67					
DB2 VERSION: V9			SCOPE: MEMBER							TO: 06/25/09 07:23:31.68					
--SUSPEND REASONS--															
----- R E S U M E R E A S O N S -----															
DATABASE	---	L O C K	R E S O U R C E	---	TOTAL	LOCAL	GLOB.	S.NFY	----	NORMAL	----	TIMEOUT/CANCEL	---	DEADLOCK	---
PAGESET	TYPE	NAME			SUSPENDS	LATCH	IRLMQ	OTHER	NMBR		AET	NMBR		AET	NMBR
TS01106	SYSLGRNG	N/P			1	1	0	0	1	0.003860	0		N/C	0	N/C
						0	0	0							
673	ROW	PAGE=X'00026A'			2	2	0	0	2	0.393065	0		N/C	0	N/C
						0	0	0							
	ROW	PAGE=X'00026B'			6	6	0	0	1	0.059315	2	19.139177	3	9.045099	
						0	0	0							
	ROW	PAGE=X'00026C'			3	3	0	0	3	7.367667	0		N/C	0	N/C
						0	0	0							
	ROW	PAGE=X'00026F'			3	3	0	0	0	N/C	1	16.617085	2	8.767127	
						0	0	0							
	** SUM OF	673		**	14	14	0	0	6	3.824741	3	18.298480	5	8.933910	
						0	0	0							
983	ROW	PAGE=X'000304'			2	2	0	0	0	N/C	2	18.534931	0		N/C
						0	0	0							
*TOTAL*															
ICMDBLO1															
						25	25	0	0	15	1.532838	5	18.393060	5	8.933910
							0	0	0						
						36	27	0	0	26	0.884413	5	18.393060	5	8.933910
							9	0	0						

The report contains a great deal of information and may look a bit confusing when you look at it the first time. Let us look a bit more carefully at the different parts that make up the report.

The left hand side contains information about the resources that experienced the suspensions, and the middle section indicates how many suspensions there were and the different types of suspensions that were incurred. Example 9-13 shows this information in more detail.

*Example 9-13 Resource and type of suspension*

DATABASE PAGESET	--- L O C K   R E S O U R C E ---		TOTAL SUSPENDS	--SUSPEND REASONS--		
	TYPE	NAME		LOCAL LATCH	GLOB. IRLMQ	S.NFY OTHER
ICMDBL01						
673	ROW	PAGE=X'00026A'	2	2	0	0
		ROW =X'01'		0	0	0

The name of the database lines up exactly under the DATABASE heading, so in our case the suspension was on an object in database ICMDBL01. The pageset or table lines up under the PAGESET heading, table 673 in our case. The lock type that got suspended is a row, and in this case, row x'01' on page x'26A' in table '673' in database ICMDBL01.

During the time the trace was active, we had two suspensions on this row (TOTAL SUSPENDS). The columns to the right in Example 9-13 give further details about the types of suspension that occurred on this row. The report distinguishes between:

- ▶ LOCAL suspensions: These are the regular local IRLM lock suspensions, for example, when two transactions want to access a resource (a row in this case) at the same time with an incompatible state (X versus S, for example).
- ▶ LATCH suspensions: These are IRLM (only) latch contentions. These occur when IRLM needs to serialize on a resource inside IRLM, for example, when IRLM has to serialize when adding or removing locks to the lock table. When that is being done, the lock table is latched for a short period of time. IRLM latch suspensions are expected to be low and of a short duration. If that is not the case, you need to perform further investigation (refer to “SUSPENSIONS (IRLM LATCH)” on page 230 for more details.)
- ▶ GLOB. suspensions; These are IRLM global lock suspensions. They only occur when running in a DB2 data sharing environment.
- ▶ IRLMQ. suspensions: These are IRLM queued requests.
- ▶ S.NFY suspensions: These are suspensions when the application has to wait for a reply from an IRLM NOTIFY request. Notices are used for many reasons to exchange information in a DB2 data sharing environment.
- ▶ OTHER suspensions. These are miscellaneous suspensions. Some of the suspensions in this category are:
  - Drain suspensions
  - Contentions with retained locks
  - Page latch suspensions

In this case, both suspensions were LOCAL IRLM lock suspensions.

The right hand side of the report (Example 9-14) contains information about what happened when the thread was resumed (when the suspension ended).

*Example 9-14 Locking resume reasons*

DATABASE PAGESET	--- L O C K R E S O U R C E ---		----- R E S U M E R E A S O N S -----					
	TYPE	NAME	NMBR	NORMAL AET	TIMEOUT/CANCEL NMBR	DEADLOCK AET	NMBR	AET
ICMDBLO1 673	ROW	PAGE=X'00026A' ROW =X'01'	2	0.393065	0	N/C	0	N/C

The report distinguishes between:

- ▶ **NORMAL resume:** This means that the transaction has been waiting for a while for the resource to become available and was able to continue processing normally once the lock became available (for example, the holder committed, the lock it was holding was freed, and the suspended transaction could continue processing).
- ▶ **TIMEOUT/CANCEL resume:** When the transaction was suspended, it got canceled or timed out (after the timeout interval had expired). In this case, the transaction was not able to continue normal processing. It either had to do a ROLLBACK, or a rollback was issued on behalf of the transaction.
- ▶ **DEADLOCK resume.** When the transaction was suspended, it was selected as the victim of a deadlock. In this case the transaction was not able to continue normal processing. It either had to do a ROLLBACK, or a rollback was issued on behalf of the transaction when it was resumed.

For all three categories, OMEGAMON PE reports the number of times as well as the average elapsed time (AET) of the suspension. In Example 9-14, the transaction was able to resume normal processing, after waiting 0.393 seconds on average.

When looking at the report in Example 9-12 on page 266, you can see that not all suspensions resulted in a normal resume. For example, for row x'01' on page x'00026B' on the same table, there were six suspensions in total: one resulted in a normal resume, two resulted in a timeout, and three resulted in a deadlock. (Note that the timeout interval on this system is much lower than the default of 60 seconds).

After all the suspensions for a particular pageset have been reported, there is a “\*\* SUM OF \*\*” row (actually two rows for every resource reported) that has the totals for this pageset.

After all suspensions for a particular database have been reported, there is a “\*\*TOTAL\*\*” with the total number of suspensions for the entire database.

At the very end of the report, there is a “\*\*GRAND TOTAL\*\*” section that summarizes all lock suspensions that were found in the trace.

## Detailed locking information

The lock suspension report allows you to identify the resources that are suspended, the type of suspension, and how long a suspension takes on average. Sometimes this information alone is not enough to understand why the suspensions occur, or why many of the suspensions result in a deadlock or timeout.

In these cases, you will have to go down to a more detailed level of tracing. In that case, you not only have to gather information about when a lock request got suspended and resumed, but trace all lock requests that are sent to IRLM. In case there are long claim or drain waits, you can also trace all of those, and whether they got suspended or not. To obtain information related to detailed locking activity, you can gather the following additional IFCIDs:

- 21** Detailed lock information. This IFCID traces each lock, unlock, change, and request sent to IRLM.
- 211** Acquiring or releasing a claim.
- 212** Acquiring or releasing a drain.

The following IFCIDs can be traced to obtain information about suspensions (locks, claims, and drains), so it is best to continue tracing them as well:

- 44 - 45** IRLM lock suspend and lock resume request.
- 213 - 214** Start/end waiting for a drain lock.
- 215 - 216** Start/end waiting for the claim count to go to zero.
- 226 - 227** Start/end page latch wait.
- 105, 107** Allow OBID translation from OBID numbers to object names.

You can use the following START TRACE command to gather this information:

```
-START TRACE(P) CLASS(30) IFCID(21,211,212,44,45,213,214,215,216,226,227)
  TDATA(CPU COR DIST) DEST(SMF/GTF)
```

Note that this trace request does not gather information about DB2 latch contention. We discuss that topic in “DB2 latch suspension detailed information” on page 279.

This trace can generate a very large number of trace records and have considerable impact on CPU processing for the transactions while the trace is active. We therefore recommend turning on the trace for a short period of time, and if possible to limit the trace command using filters such as AUTHID, PLANNAME, or some of the new filters that were introduced in DB2 9, such as PKGCOL, PKGPROG, APPNAME, WRKSTN, CONNID, and CORRID.

If you are running in a DB2 data sharing environment, you may want to add some additional IFCIDs to the trace request.

- 251** Detailed information about pageset P-lock requests. Tracing IFCID 251 can be a good alternative to tracing IFCID 21 if you are only interested in pageset P-lock requests. The processing impact of collecting IFCID 251 will be much less than when collecting IFCID 21.
- 257** If you are experiencing delays in IRLM notify processing, it is a good idea to add IFCID 257 to the trace request, as it can give you a better understanding of the notify messages being sent between the members of a data sharing group
- 259** IFCID 259 traces information about page P-lock requests. Tracing IFCID 259 can be a good alternative to tracing IFCID 21 if you are only interested in page P-lock requests. Collecting IFCID 259 will have less of an impact on processing than collecting IFCID 21.

So, in a data sharing environment, you can use these IFCIDs:

```
-START TRACE(P) CLASS(30)
  IFCID(251,257,259,21,211,212,44,45,213,214,215,216,226,227)
  TDATA(CPU COR DIST) DEST(SMF/GTF)
```

After you have gathered the trace data, you can use OMEGAMON PE to format the traces. You can use different OMEGAMON PE commands to format the trace data. Unfortunately, there is no equivalent command to the lock suspension report that we discussed in “Detailed locking information” on page 268 for detailed locking information.

The two commands that are most useful are:

```
LOCKING
  TRACE
    LEVEL(DETAIL)
```

and

```
RECTRACE
  TRACE
    LEVEL(LONG)
```

The locking trace is easier to interpret, but the RECTRACE is easier to use if you want to see locking events and, for example, the begin and end of an SQL statement combined with the locking information.

Unless you apply additional filtering using the INCLUDE and EXCLUDE keywords, both the LOCKING and RECTRACE reports will contain information about all threads that were traced. The records are presented in time sequence, so the report will have some record of thread A, followed by some of thread B, and some of thread C, before a new record for thread A appears. This is very interesting if you want to see the locking events as they occurred in the system. If you are only interested in the lock requests from a single thread, it is best to apply a filter using an INCLUDE statement, either in the GLOBAL or on the LOCKING statement. The most unique type of filtering is most likely the INSTANCE number (the middle part of the LUW-ID). So, to only look at the records for a single instance, use:

```
GLOBAL
  INCLUDE( INSTANCE ( 090625064637 ) )
```

The instance number is present in all thread related trace records. You can pick the instance number from an accounting record when you see that this transaction was suffering from a long lock/latch suspension, and use it as a filter to obtain only the lock requests from that transaction.



## LOCKING TRACE LEVEL(DETAIL)

Example 9-15 is an excerpt from a LOCKING TRACE LEVEL(DETAIL) report. In a locking trace, you will only find LOCK, UNLOCK, and CHANGE requests, as well as deadlock and timeout records (if they are present in the trace). In other words, a locking trace only reports lock related events. By looking at a locking trace, you cannot tell which SQL statement was running when the lock request was issued. Depending on the problem you are analyzing, this can be important or irrelevant. If you want to see SQL and locking events in the same report, it is probably easiest to use a RECTRACE report, as shown in Example 9-16 on page 273.

Example 9-15 LOCKING TRACE LEVEL(DETAIL) report

---

LOCATION: WTSC03	OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)	PAGE: 1-686
GROUP: N/P	LOCKING TRACE - DETAIL	REQUESTED FROM: ALL 07:22:11.00
MEMBER: N/P		TO: DATES 07:23:12.00
SUBSYSTEM: DB9A		ACTUAL FROM: 06/25/09 07:22:42.64
DB2 VERSION: V9	SCOPE: MEMBER	PAGE DATE: 06/25/09

---

PRIMAUTH CORRNAME CONNTYPE	EVENT TIMESTAMP	---	LOCK	RESOURCE	---	EVENT SPECIFIC DATA
ORIGAUTH CORRNMBR INSTANCE	RELATED TIMESTAMP	EVENT	TYPE	NAME		
PLANNAME CONNECT						
ADMIN001 java DRDA	07:23:11.79547427	LOCK	SKPT	COLL=N/C	DURATION=COMMIT	STATE=S
ADMIN001 'BLANK' 090625064637		REQUEST		PKID=N/C	RSN CODE= 0	RTNCD= 0
DISTSERV SERVER				CTKN=181F9E970F989618	HASH	=X'00000380'
REQLOC :::10.225.148.139						
ENDUSER :ADMIN001						
WSNAME :dbcmadmi01.rze.d						
TRANSACT:java						
	07:23:11.79560857	LOCK	PAGESET	DB =ICMDBL01	DURATION=COMMIT	STATE=IX
		REQUEST		OB =ITEELSTS	RSN CODE= 0	RTNCD= 0
					HASH	=X'00015478'
	07:23:11.79562086	LOCK	TABLE	DB =ICMDBL01	DURATION=COMMIT	STATE=IX
		REQUEST		OB =424	RSN CODE= 0	RTNCD= 0
					HASH	=X'0101A878'
	07:23:11.79575019	LOCK	ROW	DB =ICMDBL01	DURATION=COMMIT	STATE=X
		REQUEST		OB =424	RSN CODE= 0	RTNCD= 0
				PAGE=X'000309'	HASH	=X'2495BB09'
				ROW =X'13'		
	07:23:11.79596480	CHANGE	PAGESET	DB =ICMDBL01	DURATION=COMMIT	STATE=IX
N/P		REQUEST		OB =IT11LSTS	RSN CODE= 0	RTNCD= 0
					HASH	=X'00014678'
	07:23:11.79598321	CHANGE	TABLE	DB =ICMDBL01	DURATION=COMMIT	STATE=IX
N/P		REQUEST		OB =366	RSN CODE= 0	RTNCD= 0
					HASH	=X'01016E78'
	07:23:11.79609594	LOCK	ROW	DB =ICMDBL01	DURATION=MANUAL	STATE=U
		REQUEST		OB =366	RSN CODE= 0	RTNCD= 0
				PAGE=X'000091'	HASH	=X'4453D891'
				ROW =X'18'		
	07:23:11.79611941	CHANGE	ROW	DB =ICMDBL01	DURATION=COMMIT	STATE=X
	07:23:11.79609594	REQUEST		OB =366	RSN CODE= 0	RTNCD= 0
				PAGE=X'000091'	HASH	=X'4453D891'
				ROW =X'18'		

---

As with all OMEGAMON PE reports, the thread identification information is on the left hand side of the output.

```
PRIMAUTH: ADMIN001
CORRNAME: java
CONNTYPE: DRDA
ORIGAUTH: ADMIN001
CORRNMBR: 'BLANK'
INSTANCE: 090625064637
PLANNAME: DISTSERV
CONNECT : SERVER
REQLOC  : :::10.225.148.139
ENDUSER : ADMIN001
WSNAME  : dbcmadmi01.rze.d
TRANSACT: java
```

We have the time stamp at which the event occurred. For a locking request that did not get suspended, this is the time the locking request was granted. When the request is suspended, it is the time the lock request was suspended. When the lock request is resumed, it is the time stamp of the lock request that was resumed. For easy reference, on a lock resume request, OMEGAMON PE provides the time stamp for when the request was suspended, so you can see how long it was suspended or go back in time to find the matching suspension record.

In the middle, we have the type of lock that was requested, such as PAGESET, TABLE, or ROW, and the resource for which the lock was requested.

On the right hand side, there is information about the state at which the lock was requested, the duration, the IRLM return code, and the IRLM reason code. An explanation of all the IRLM return and reason codes can be found in *IRLM Messages and Codes for IMS and DB2 for z/OS*, GC19-2666.

For example, the following is a IX lock request that will be held until commit (RELEASE(COMMIT)) for pageset ICMDBL01.ITEELSTS. The request was granted, and ends with return code and reason code zero:

```
LOCK      PAGESET  DB  =ICMDBL01      DURATION=COMMIT  STATE=IX
REQUEST   OB      =ITEELSTS          RSN CODE= 0      RTNCD= 0
                                         HASH      =X'00015478'
```

In Example 9-15 on page 271, we first get a lock on the skeleton package table, then acquire an IX pageset lock, and an IX table lock, and an X lock on a row. Then we acquire an IX lock on another pageset, and table, and a U lock on a row. We promote (change) that U lock into an X lock. Note that the CHANGE request has two timestamps:

```
07:23:11.79611941 CHANGE  ROW      DB  =ICMDBL01
07:23:11.79609594 REQUEST OB      =366
                                         PAGE=X'000091'
                                         ROW  =X'18'
```

The first one (07:23:11.79611941) is the time that the CHANGE request is granted. The second one (07:23:11.79609594) is the time that the initial lock request was granted. Having both time stamps makes it easy to go back to the trace to see when that happened and also to quickly see how long we have already been holding on to that lock.

## RETRACE TRACE LEVEL(LONG)

Example 9-16 shows the same time frame as Example 9-15 on page 271, but formatted using the RETRACE TRACE LEVEL(LONG) command. Note that we did not limit the RETRACE output to just the IFCIDs that are related to locking; you can also see the package allocation and start and end of each SQL statement.

Example 9-16 RETRACE TRACE LEVEL(LONG) report

```
LOCATION: WTSC03                      OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)          PAGE: 1-8034
GROUP: N/P                          RECORD TRACE - LONG                      REQUESTED FROM: ALL      07:22:11.00
MEMBER: N/P                          TO: DATES      07:23:12.00
SUBSYSTEM: DB9A                     ACTUAL FROM: 06/25/09 07:22:42.64
DB2 VERSION: V9                     PAGE DATE: 06/25/09
PRIMAUTH CONNECT                     INSTANCE      END_USER      WS_NAME      TRANSACT
ORIGAUTH CORRNAME CONNTYPE          RECORD TIME  DESTNO ACE IFC DESCRIPTION DATA
PLANNAME CORRNMBR          TCB CPU TIME          ID

-----
ADMIN001 SERVER 090625064637 ADMIN001 dbcmadmi01.rze.d java
ADMIN001 java DRDA 07:23:11.79547427 307216 1 21 LOCK DETAIL EQ: 4
DISTSERV 'BLANK' N/P                                         NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
                                                                REQUESTING LOCATION: ::10.225.148.139
                                                                REQUESTING TIMESTAMP: N/P
                                                                AR NAME: dbcmadmi01.rze.d PRDID: CLNT/SER V8 R2 M9
                                                                ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'

-----
LOCK RES TYPE: SKELETON PACKAGE TABLE LOCK COLL: N/C PKID: N/C CTKN: X'181F9E970F989618'
IRLM FUNC CODE : LOCK (NAME) RETURN TOKEN: X'7F547230' REQUEST TOKEN : X'00000000'
LOCK STATE : SHARED DB2 TOKEN : X'00C22000155C6428' IRLM RETURN CODE : 0
LOCK ATTRIBUTES: NMODIFY NOFORCE PROP TO XES : NO ASYN TO XES : NO
LOCK DURATION : COMMIT REQUEST TYPE: IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN : X'00000000' GLOBAL/LOCAL: GLOBAL OWNER : 'BLANK'
CACHED STATE : N/A LOCK HASH VALUE : X'00000380'
QW0021CL: X'00' QW0021U : X'015F0093143D93E8' QW0021CT: X'00000000' QW0021FL: B'10010000'
QW0021F3: B'00000000' QW0021O : X'015F0093143D9340' QW0021IR: X'0000' QW0021F2: B'00000000'

-----
07:23:11.79552666 307217 1 177 PACKAGE NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
N/P ALLOCATION REQUESTING LOCATION: ::10.225.148.139
                                                                REQUESTING TIMESTAMP: N/P
                                                                AR NAME: dbcmadmi01.rze.d PRDID: CLNT/SER V8 R2 M9
                                                                ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
                                                                LOCATION : WTSC03
                                                                COLLECTION ID : ICMCOL
                                                                PACKAGE ID : ICMPLSCM
                                                                CONSISTENCY TOKEN: X'181F9E970F989618'
                                                                VERSION NAME : N/P
                                                                DYNAMICRULES : RUN
                                                                PLAN : DISTSERV
                                                                ISOLATION : CS
                                                                ACQUIRE : USE
                                                                RELEASE : COMMIT
                                                                REOPTIMIZATION : NO
                                                                DEFERPREPARE : NO
                                                                KEEPYNAMIC : NO
                                                                DBPROTOCOL : DRDA
                                                                OPT_HINT_IDENT : 'BLANK'
                                                                IMMEDIATEWRITE : NO

07:23:11.79558333 307218 1 61 INSERT --> java
N/P START NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
                                                                REQUESTING LOCATION: ::10.225.148.139
                                                                REQUESTING TIMESTAMP: N/P
                                                                AR NAME: dbcmadmi01.rze.d PRDID: CLNT/SER V8 R2 M9
                                                                ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
                                                                LOCATION NAME : WTSC03
                                                                PKG COLLECTION ID : ICMCOL
                                                                PROGRAM NAME : ICMPLSCM
                                                                CONSISTENCY TOKEN : X'181F9E970F989618'
                                                                STATEMENT NUMBER : 13852
                                                                STATEMENT TYPE : INSERT TYPE
                                                                CURSOR NAME : N/P
                                                                QUERY COMMAND ID : N/P
                                                                QUERY INSTANCE ID : N/P
                                                                ISOLATION : CS
                                                                REOPTIMIZATION : NO

07:23:11.79560857 307219 1 21 LOCK DETAIL NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
N/P REQUESTING LOCATION: ::10.225.148.139
                                                                REQUESTING TIMESTAMP: N/P
                                                                AR NAME: dbcmadmi01.rze.d PRDID: CLNT/SER V8 R2 M9
                                                                ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'

-----
LOCK RES TYPE: PAGESET LOCK DBID: ICMDBL01 OBID: ITEELSTS
```

```

IRLM FUNC CODE : LOCK (NAME)          RETURN TOKEN: X'7F54ABA0'          REQUEST TOKEN      : X'00000000'
LOCK STATE      : INTENT EXCLUSIVE    DB2 TOKEN      : X'00C22000155C6428'    IRLM RETURN CODE   : 0
LOCK ATTRIBUTES: MODIFY NOFORCE       PROP TO XES    : NO                      ASYN TO XES        : NO
LOCK DURATION   : COMMIT              REQUEST TYPE:  IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN    : X'00000000'         GLOBAL/LOCAL: GLOBAL                     OWNER              : 'BLANK'
CACHED STATE    : N/A                 LOCK HASH VALUE : X'00015478'
QW0021CL: X'00'          QW0021U : X'015F009300000000' QW0021CT: X'7BCC4B00' QW0021FL: B'00110010'
QW0021F3: B'00000000'   QW0021O : X'015F0093143D9340' QW0021IR: X'0000'   QW0021F2: B'00000000'
-----
07:23:11.79562086 307220 1 21 LOCK DETAIL java
N/P
NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
REQUESTING LOCATION: ::10.225.148.139
REQUESTING TIMESTAMP: N/P
AR NAME: dbcmadmi01.rze.d PRDID: CLNT/SER V8 R2 M9
ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
-----
LOCK RES TYPE: TABLE LOCK          DBID: ICMDBL01          OBID: 424
IRLM FUNC CODE : LOCK (NAME)          RETURN TOKEN: X'7F001FD0'          REQUEST TOKEN      : X'00000000'
LOCK STATE      : INTENT EXCLUSIVE    DB2 TOKEN      : X'00C22000155C6428'    IRLM RETURN CODE   : 0
LOCK ATTRIBUTES: MODIFY NOFORCE       PROP TO XES    : NO                      ASYN TO XES        : NO
LOCK DURATION   : COMMIT              REQUEST TYPE:  IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN    : X'00000000'         GLOBAL/LOCAL: GLOBAL                     OWNER              : 'BLANK'
CACHED STATE    : N/A                 LOCK HASH VALUE : X'0101A878'
QW0021CL: X'00'          QW0021U : X'015F009300000000' QW0021CT: X'7BCC4B00' QW0021FL: B'00110010'
QW0021F3: B'00000000'   QW0021O : X'015F0093143D9340' QW0021IR: X'0000'   QW0021F2: B'00000000'
-----
07:23:11.79575019 307221 1 21 LOCK DETAIL NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
N/P
REQUESTING LOCATION: ::10.225.148.139
REQUESTING TIMESTAMP: N/P
AR NAME: dbcmadmi01.rze.d PRDID: CLNT/SER V8 R2 M9
ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
-----
LOCK RES TYPE: ROW LOCK          DBID: ICMDBL01          OBID: 424 RESOURCE ID: X'00030913'
IRLM FUNC CODE : LOCK (NAME)          RETURN TOKEN: X'7F5450C0'          REQUEST TOKEN      : X'00000000'
LOCK STATE      : EXCLUSIVE           DB2 TOKEN      : X'00C22000155C6428'    IRLM RETURN CODE   : 0
LOCK ATTRIBUTES: MODIFY NOFORCE       PROP TO XES    : NO                      ASYN TO XES        : NO
LOCK DURATION   : COMMIT              REQUEST TYPE:  IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN    : X'7F54ABA0'         GLOBAL/LOCAL: GLOBAL                     OWNER              : 'BLANK'
CACHED STATE    : N/A                 LOCK HASH VALUE : X'2495BB09'
QW0021CL: X'00'          QW0021U : X'015F009300000000' QW0021CT: X'7BCC4B00' QW0021FL: B'010110010'
QW0021F3: B'00000000'   QW0021O : X'015F0093143D9340' QW0021IR: X'0000'   QW0021F2: B'00000000'
-----
07:23:11.79578367 307222 1 58 END SQL <-- java
N/P
NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
REQUESTING LOCATION: ::10.225.148.139
REQUESTING TIMESTAMP: N/P
AR NAME: dbcmadmi01.rze.d PRDID: CLNT/SER V8 R2 M9
ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
-----
LOCATION NAME      : WTSC03
PKG COLLECTION ID : ICMCOL
PROGRAM NAME     : ICMPLSCM
CONSISTENCY TOKEN : X'181F9E970F989618'
STATEMENT NUMBER : 13852 QUERY COMMAND ID : N/P QUERY INSTANCE ID: N/P
SQLCAID: SQLCA SQLCABC 136 SQLCODE : 0 SQLSTATE: 00000
SQLERRD1 0 SQLERRD2 0 SQLERRD3 1 SQLERRP : DSN SQLLEXT : 00000
SQLERRD4 -1 SQLERRD5 0 SQLERRD6 0 SQLWARN0: SQLWARN1: SQLWARN2: SQLWARN3:
SQLERRD8 SQLERRD9 SQLERRD10 SQLERRD11 SQLERRD12 SQLERRD13 SQLERRD14 SQLERRD15 SQLERRD16 SQLERRD17 SQLERRD18 SQLERRD19 SQLERRD20 SQLERRD21 SQLERRD22 SQLERRD23 SQLERRD24 SQLERRD25 SQLERRD26 SQLERRD27 SQLERRD28 SQLERRD29 SQLERRD30 SQLERRD31 SQLERRD32 SQLERRD33 SQLERRD34 SQLERRD35 SQLERRD36 SQLERRD37 SQLERRD38 SQLERRD39 SQLERRD40 SQLERRD41 SQLERRD42 SQLERRD43 SQLERRD44 SQLERRD45 SQLERRD46 SQLERRD47 SQLERRD48 SQLERRD49 SQLERRD50 SQLERRD51 SQLERRD52 SQLERRD53 SQLERRD54 SQLERRD55 SQLERRD56 SQLERRD57 SQLERRD58 SQLERRD59 SQLERRD60 SQLERRD61 SQLERRD62 SQLERRD63 SQLERRD64 SQLERRD65 SQLERRD66 SQLERRD67 SQLERRD68 SQLERRD69 SQLERRD70 SQLERRD71 SQLERRD72 SQLERRD73 SQLERRD74 SQLERRD75 SQLERRD76 SQLERRD77 SQLERRD78 SQLERRD79 SQLERRD80 SQLERRD81 SQLERRD82 SQLERRD83 SQLERRD84 SQLERRD85 SQLERRD86 SQLERRD87 SQLERRD88 SQLERRD89 SQLERRD90 SQLERRD91 SQLERRD92 SQLERRD93 SQLERRD94 SQLERRD95 SQLERRD96 SQLERRD97 SQLERRD98 SQLERRD99 SQLERRD100
SQLERRM:
DATA TYPE SEQD ROW PROC 0 ROW EXAM 0 STG1-QUAL 0 STG2-QUAL 0 ROW INSRT 1
ROW UPDTE 0 ROW DELET 0 PAGES 2 RI SCAN 0 RI DELET 0
LOB SCAN 0 LOB UPDTE 0
-----
07:23:11.79593583 307223 1 61 UPDATE --> NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
N/P START
REQUESTING LOCATION: ::10.225.148.139
REQUESTING TIMESTAMP: N/P
AR NAME: dbcmadmi01.rze.d PRDID: CLNT/SER V8 R2 M9
ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
LOCATION NAME : WTSC03
PKG COLLECTION ID : ICMCOL
PROGRAM NAME : ICMPLSRI
CONSISTENCY TOKEN : X'184522150A4772BA'
STATEMENT NUMBER : 14178
STATEMENT TYPE : UPDATE TYPE- NON CURSOR
CURSOR NAME : N/P
QUERY COMMAND ID : N/P
QUERY INSTANCE ID : N/P
ISOLATION : CS
REOPTIMIZATION : NO
07:23:11.79596480 307224 1 21 LOCK DETAIL java
N/P
NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4

```

```

REQUESTING LOCATION: ::10.225.148.139
REQUESTING TIMESTAMP: N/P
AR NAME: dbcmadm01.rze.d      PRDID: CLNT/SER V8 R2 M9
ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
-----
LOCK RES TYPE: PAGESET LOCK          DBID: ICMDBL01          OBID: IT11LSTS
IRLM FUNC CODE : CHANGE (NAME)      RETURN TOKEN: X'7F5568C0'    REQUEST TOKEN   : X'00000000'
LOCK STATE      : INTENT EXCLUSIVE   DB2 TOKEN       : X'00C22000155C6428' IRLM RETURN CODE : 0
LOCK ATTRIBUTES: MODIFY NOFORCE      PROP TO XES : NO      ASYN TO XES      : NO
LOCK DURATION   : COMMIT             REQUEST TYPE:    IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN    : X'00000000'        GLOBAL/LOCAL: GLOBAL    OWNER           : 'BLANK'
CACHED STATE    : N/A                LOCK HASH VALUE : X'00014678'
QW0021CL: X'00'      QW0021U : X'015F0093143D93E8' QW0021CT: X'7BCC4B00' QW0021FL: B'00110010'
QW0021F3: B'00000000' QW0021O : X'015F0093143D9340' QW0021IR: X'0000' QW0021F2: B'10000000'
-----
07:23:11.79598321 307225 1 21 LOCK DETAIL NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
N/P REQUESTING LOCATION: ::10.225.148.139
REQUESTING TIMESTAMP: N/P
AR NAME: dbcmadm01.rze.d      PRDID: CLNT/SER V8 R2 M9
ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
-----
LOCK RES TYPE: TABLE LOCK          DBID: ICMDBL01          OBID: 366
IRLM FUNC CODE : CHANGE (NAME)      RETURN TOKEN: X'7F556F60'    REQUEST TOKEN   : X'00000000'
LOCK STATE      : INTENT EXCLUSIVE   DB2 TOKEN       : X'00C22000155C6428' IRLM RETURN CODE : 0
LOCK ATTRIBUTES: MODIFY NOFORCE      PROP TO XES : NO      ASYN TO XES      : NO
LOCK DURATION   : COMMIT             REQUEST TYPE:    IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN    : X'00000000'        GLOBAL/LOCAL: GLOBAL    OWNER           : 'BLANK'
CACHED STATE    : N/A                LOCK HASH VALUE : X'01016E78'
QW0021CL: X'00'      QW0021U : X'015F009300000000' QW0021CT: X'7BCC4B00' QW0021FL: B'00110010'
QW0021F3: B'00000000' QW0021O : X'015F0093143D9340' QW0021IR: X'0000' QW0021F2: B'00000000'
-----
07:23:11.79609594 307227 1 21 LOCK DETAIL java
N/P NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
REQUESTING LOCATION: ::10.225.148.139
REQUESTING TIMESTAMP: N/P
AR NAME: dbcmadm01.rze.d      PRDID: CLNT/SER V8 R2 M9
ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
-----
LOCK RES TYPE: ROW LOCK              DBID: ICMDBL01          OBID: 366 RESOURCE ID: X'00009118'
IRLM FUNC CODE : LOCK (NAME)         RETURN TOKEN: X'7F002420'    REQUEST TOKEN   : X'00000000'
LOCK STATE      : UPDATE              DB2 TOKEN       : X'00C22000155C6428' IRLM RETURN CODE : 0
LOCK ATTRIBUTES: NMODIFY NOFORCE      PROP TO XES : NO      ASYN TO XES      : NO
LOCK DURATION   : MANUAL              REQUEST TYPE:    IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN    : X'7F5568C0'        GLOBAL/LOCAL: GLOBAL    OWNER           : 'BLANK'
CACHED STATE    : N/A                LOCK HASH VALUE : X'4453D891'
QW0021CL: X'00'      QW0021U : X'015F0093143D93E8' QW0021CT: X'7BCC4B00' QW0021FL: B'00110000'
QW0021F3: B'00000000' QW0021O : X'015F0093143D9340' QW0021IR: X'0000' QW0021F2: B'00000000'
-----
07:23:11.79611941 307228 1 21 LOCK DETAIL NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
N/P REQUESTING LOCATION: ::10.225.148.139
REQUESTING TIMESTAMP: N/P
AR NAME: dbcmadm01.rze.d      PRDID: CLNT/SER V8 R2 M9
ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
-----
LOCK RES TYPE: N/P                   NAME: N/P
IRLM FUNC CODE : CHANGE (TOKEN)      RETURN TOKEN: X'7F002420'    REQUEST TOKEN   : X'7F002420'
LOCK STATE      : EXCLUSIVE           DB2 TOKEN       : X'00C22000155C6428' IRLM RETURN CODE : 0
LOCK ATTRIBUTES: MODIFY NOFORCE      PROP TO XES : NO      ASYN TO XES      : NO
LOCK DURATION   : COMMIT              REQUEST TYPE:    IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN    : X'7F5568C0'        GLOBAL/LOCAL: GLOBAL    OWNER           : 'BLANK'
CACHED STATE    : N/A                LOCK HASH VALUE : X'4453D891'
QW0021CL: X'00'      QW0021U : X'015F009300000000' QW0021CT: X'7BCC4B00' QW0021FL: B'00110010'
QW0021F3: B'00000000' QW0021O : X'015F0093143D9340' QW0021IR: X'0000' QW0021F2: B'00010000'
-----
07:23:11.79639314 307229 1 58 END SQL <-- java
N/P NETWORKID: GAE1948B LUNAME: C3F5 LUWSEQ: 4
REQUESTING LOCATION: ::10.225.148.139
REQUESTING TIMESTAMP: N/P
AR NAME: dbcmadm01.rze.d      PRDID: CLNT/SER V8 R2 M9
ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
-----
LOCATION NAME      : WTSC03
PKG COLLECTION ID : ICMCOL
PROGRAM NAME      : ICMPLSRI
CONSISTENCY TOKEN : X'184522150A4772BA'
STATEMENT NUMBER  : 14178 QUERY COMMAND ID : N/P QUERY INSTANCE ID: N/P
SQLCAID: SQLCA    SQLCABC 136
SQLERRD1 0 SQLERRD2 0 SQLERRD3 1 SQLERRP : DSN SQLSTATE: 00000
SQLERRD4 -1 SQLERRD5 0 SQLERRD6 0 SQLWARN0: SQLWARN1: SQLWARN2: SQLWARN3:
SQLERRM:          SQLWARN4: SQLWARN5: SQLWARN6: SQLWARN7:
          SQLWARN8: SQLWARN9: SQLWARNA:
-----

```

DATA TYPE	INDX ROW PROC	5 ROW EXAM	0 STG1-QUAL	0 STG2-QUAL	0 ROW INSRT	0
ROW UPDTE	0 ROW DELET	3 PAGES	13 RI SCAN	2 RI DELET	0	
LOB SCAN	0 LOB UPDTE	0				
DATA TYPE	SEQD ROW PROC	1 ROW EXAM	0 STG1-QUAL	1 STG2-QUAL	1 ROW INSRT	0
ROW UPDTE	1 ROW DELET	0 PAGES	1 RI SCAN	0 RI DELET	0	
LOB SCAN	0 LOB UPDTE	0				

With a RECTRACE, it is easier to understand which lock request was issued by which SQL statement, but the RECTRACE has a tendency to produce an enormous amount of output, so you should always apply as much filtering (using INCLUDE and EXCLUDE) as possible, as well as using FROM and TO time stamps to limit the scope of the trace.

Comparing the information from the LOCKING trace and the RECTRACE for the same lock request, as shown in Example 9-17, you see that the RECTRACE contains additional fields. In most cases, the information from the LOCKING trace is sufficient to get a good picture of the locking behavior of an application. However, it is good to keep in mind that, if needed, you can still go down one more level of detail and use the RECTRACE instead of the LOCKING trace.

#### Example 9-17 Comparing LOCKING and RECTRACE

##### LOCKING TRACE

```
07:23:11.79560857 LOCK    PAGESET  DB  =ICMDBL01    DURATION=COMMIT  STATE=IX
                    REQUEST      OB  =ITEELSTS    RSN CODE= 0      RTNCD= 0
                                         HASH    =X'00015478'
```

##### RECTRACE

```
07:23:11.79560857 307219 1 21 LOCK DETAIL  NETWORKID: GAE1948B LUNAME: C3F5 LUVSEQ: 4
                    N/P      REQUESTING LOCATION: ::10.225.148.139
                                REQUESTING TIMESTAMP: N/P
                                AR NAME: dbcmadm01.rze.d PRDID: CLNT/SER V8 R2 M9
                                ACCTKN X'C7C1C5F1F9F4F8C24BC3F3C6F5090625064637404040'
```

LOCK RES TYPE: PAGESET LOCK	DBID: ICMDBL01	OBID: ITEELSTS
IRLM FUNC CODE : LOCK (NAME)	RETURN TOKEN: X'7F54ABA0'	REQUEST TOKEN : X'00000000'
LOCK STATE : INTENT EXCLUSIVE	DB2 TOKEN : X'00C22000155C6428'	IRLM RETURN CODE : 0
LOCK ATTRIBUTES: MODIFY NOFORCE	PROP TO XES : NO	ASYN TO XES : NO
LOCK DURATION : COMMIT	REQUEST TYPE:	IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN : X'00000000'	GLOBAL/LOCAL: GLOBAL	OWNER : 'BLANK'
CACHED STATE : N/A		LOCK HASH VALUE : X'00015478'
QW0021CL: X'00'	QW0021U : X'015F009300000000'	QW0021CT: X'7BCC4B00'
	QW00210 : X'015F0093143D9340'	QW0021FL: B'00110010'
QW0021F3: B'00000000'	QW0021R: X'0000'	QW0021F2: B'00000000'

An explanation of all the fields of IFCID 21 (and all other DB2 trace records) can be found in the hlq.SDSNIVPD(DSNWMSGs) data set, where hlq is the high level qualifier that you use for your DB2 system data sets.

### Information about DB2 lock avoidance

Lock avoidance is a very powerful mechanism to reduce the number of locks that have to be acquired by a transaction. However, it is not always easy to assess whether lock avoidance is being exploited in a specific environment. In “UNLOCK REQUESTS” on page 232, we described a very high level technique to get some idea about whether or not lock avoidance is effective, but sometimes you need a more detailed view. Again, DB2 traces are useful to solve this problem. The following IFCIDs can be traced to gather information about lock avoidance:

#### 218

Lock avoidance summary information. This trace record indicates whether lock avoidance techniques have been used by a unit of work in general, and for each object (pageset), whether or not lock avoidance was used.

223

Detailed information about lock avoidance techniques. If this IFCID is active, a trace record is written each time lock avoidance is use. If lock avoidance was used on 100 pages, 100 trace records are produced. If lock avoidance is doing a good job, tracing this IFCID can produce a very large number of trace records and will impact the CPU time of the transaction.

You can use the following START TRACE command to gather this information:

```
-START TRACE(P) CLASS(30) IFCID(218,223)
  TDATA(CPU COR DIST) DEST(SMF/GTF)
```

Example 9-18 shows the formatted IFCID 218 trace record using the OMEGAMON PE RECTRACE TRACE LEVEL(LONG) command.

Example 9-18 Formatted IFCID 218 record

LOCATION: DB9A			OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)					
GROUP: N/P			RECORD TRACE - LONG					
MEMBER: N/P								
SUBSYSTEM: DB9A								
DB2 VERSION: V9								
PRIMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME				TRANSACT
ORIGAUTH	CORRNAME	CONNTYPE	RECORD TIME	DESTNO ACE	IFC	DESCRIPTION	DATA	
PLANNAME	CORRNMBR		TCB CPU TIME		ID			
-----								
BART	DB2CALL	C495554ECF50	'BLANK'	'BLANK'				'BLANK'
BART	BARTSQR3	DB2CALL	19:24:13.64536525	818929	2	218	LOCK AVOIDANCE	
DSNREXX	'BLANK'		0.00012412					SUMMARY
-----								
LOCK AVOID DURING UNIT OF WORK: YES				NO. PAGE SET SUBRECORDS:			11	
DBID: 754		PSID: 35	LOCK AVOID DURING UNIT OF WORK: NO					
DBID: 754		PSID: 40	LOCK AVOID DURING UNIT OF WORK: YES					
DBID: 754		PSID: 18	LOCK AVOID DURING UNIT OF WORK: YES					
DBID: 754		PSID: 2	LOCK AVOID DURING UNIT OF WORK: NO					
DBID: 754		PSID: 30	LOCK AVOID DURING UNIT OF WORK: YES					
DBID: 6		PSID: 753	LOCK AVOID DURING UNIT OF WORK: YES					
DBID: 754		PSID: 9	LOCK AVOID DURING UNIT OF WORK: NO					
DBID: 754		PSID: 63	LOCK AVOID DURING UNIT OF WORK: NO					
DBID: 6		PSID: 371	LOCK AVOID DURING UNIT OF WORK: NO					
DBID: 1		PSID: 127	LOCK AVOID DURING UNIT OF WORK: NO					
DBID: 0		PSID: 754	LOCK AVOID DURING UNIT OF WORK: NO					
-----								

The example shows that lock avoidance for access to data was used by this unit of work, and then it has 11 page sets that were accessed by this unit of work, and an indication whether or not lock avoidance was used on that object.

In Example 9-18 on page 277, OMEGAMON PE was not able to do the DBID/PSID translation, so you may have to do a catalog query to discover which objects are involved, as shown in Example 9-19, where we retrieve the information for one of the objects in the list.

Example 9-19 List database and table space name

```

SELECT  DBID      , PSID
        , DBNAME  , NAME AS TSNAME
        , SEGSIZE , PARTITIONS , TYPE
FROM SYSIBM.SYSTABLESPACE
WHERE DBID= 754 AND PSID= 18 ;

```

DBID	PSID	DBNAME	TSNAME	SEGSIZE	PARTITIONS	TYPE
754	18	GLWBSQLP	GLWSEPA	64	0	

Example 9-18 on page 277 shows the lock avoidance summary record of a unit of work. Example 9-20 shows the formatted lock avoidance detailed record (IFCID 223) from the same unit of work for one of the times where we successfully used lock avoidance for either qualifying or non-qualifying rows on the object shown in bold in Example 9-18 on page 277.

Example 9-20 Formatted IFCID 223 record

LOCATION: DB9A		OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)					PAGE: 1-12704		
GROUP: N/P		RECORD TRACE - LONG					REQUESTED FROM: ALL		19:24:13.25
MEMBER: N/P							TO: DATES		19:24:13.65
SUBSYSTEM: DB9A							ACTUAL FROM: 08/03/09		19:24:13.25
DB2 VERSION: V9							PAGE DATE: 08/03/09		
PRIMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME		TRANSACTION			
ORIGAUTH	CORRNAME	CONNTYPE	RECORD TIME	DESTNO ACE	IFC	DESCRIPTION	DATA		
PLANNAME	CORRNMBR		TCB CPU TIME		ID				
-----									
BART	DB2CALL	C495554ECF50	'BLANK'	'BLANK'			'BLANK'		
BART	BARTSQR3	DB2CALL	19:24:13.64004436	818901	1 223	LOCK AVOIDANCE	NETWORKID: USIBMSC	LUNAME: SCPDB9A	LUWSEQ: 1
DSNREXX	'BLANK'		29.21877933			DETAIL	LOCK RES TYPE: DATA PAGE		
							DBID: 754	OBID: 19	
							TABLE_SPACE_TYPE: N		
							RESOURCE ID: X'001C8200'		
							QW0223U X'00A9008420A28540'		
							QW0223O X'00A9008420A28498'		
							QW0223CL X'00'		

Note that IFCID 223 shows OBID 19 is involved, while IFCID 218 shows PSID 20 is doing lock avoidance. OBID 19 is a table inside the segmented table space with PSID 20. Example 9-21 shows the catalog information for OBID 19, and you can see that the GLWTEPA (OBID 19) is in table space GLWSEPA (PSID 20).

Example 9-21 List owner and table name

```

SELECT  DBID      , OBID
        , SUBSTR(CREATOR,1,15) AS OWNER
        , SUBSTR(NAME,1,15) AS TAB_NAME
        , DBNAME  , TSNAME
FROM SYSIBM.SYSTABLES
WHERE DBID= 754 AND OBID= 19 ;

```

DBID	OBID	OWNER	TAB_NAME	DBNAME	TSNAME
754	19	GLWBSQLP	GLWTEPA	GLWBSQLP	GLWSEPA

DB2 managed to do lock avoidance on the GLWTEPA table for page x'001C82'. The last byte x'00 'is only used when row level locking is in effect. QW0223CL X'00' indicates that lock avoidance is in place.



## DB2 latch suspension detailed information

If the DB2 statistics latch contention section indicates that one or more of the latch classes shows a high (greater than 1-10 KBps) latch contention and it is not immediately clear which of the underlying latches is really responsible, you may have to start a more detailed trace to identify the actual latch type that is causing the high latch contention.

You can use the following IFCIDs to obtain more detailed information about DB2 latch suspensions:

<b>51-52</b>	Shared latch resume and shared latch wait
<b>56-57</b>	Exclusive latch wait and exclusive latch resume

These IFCIDs have the real DB2 latch number and the address of the latch. If the system experiences a high degree of DB2 latch contention, and if you want to identify the actual latches that are triggering the latch contention, it is normally sufficient to trace only IFCID 56 and 57. As there are many latch contentions, there must be X latch requests that will be suspended (since S and S are compatible), so in order to reduce the amount of trace data being gathered, there is normally no need to trace IFCID 51 and 52.

The following trace command will gather this information:

```
-STA TRA(P) CLASS(30) IFCID(56,57) DEST(SMF/GTF) TDATA(CPU COR DIST)
```

This trace can generate a very large number of trace records and generate a considerable amount of CPU processing for the transactions while the trace is active. We recommend turning on tracing for a short period of time (5 to 10 seconds is normally enough).

Note that there is no equivalent trace record for IFCID 21 (lock detail information) when dealing with DB2 latches. DB2 only has the ability to generate a trace record when a latch request runs into contention, so you cannot trace a latch request that is granted immediately.

## DB2 page latch suspension

Obtaining a page latch suspension report is discussed in “Lock suspension information” on page 265. The OMEGAMON PE lock suspension report shows page latch wait suspensions as “REASON OTHER” suspensions. The lock suspension report is very useful when you need to determine which objects and pages are causing most of the page latch contention.

Sometimes it is necessary to analyze individual page latch suspension. This is most often the case when you are dealing with very few, but very long, page latch suspension. In that case, you need to look at the individual page latch suspensions and construct the picture of what was going on at that time to determine the root cause of the problem.

You can use the following START TRACE command to gather this information:

```
-START TRACE(P) CLASS(30) IFCID(226,227) TDATA(CPU COR DIST) DEST(SMF/GTF)
```

This trace can generate a very large number of trace records and generate a considerable CPU processing impact for the transactions while the trace is active. We recommend turning on the trace for a short period of time, and, if possible, to limit the trace command using filters such as AUTHID, PLANNAME, or some of the new filters that were introduced in DB2 9, such as PKGCOL, PKGPROG, APPNAME, WRKSTN, CONNID, and CORRID.

This trace only gathers information about individual page latch contention. To get a better idea of what is going on, it is often necessary to trace additional information in addition to IFCIDs 226 and 227. Which additional information you need depends on the type of page latch suspension which you are dealing with. Often, a performance trace class 1, 2, or 3 or an I/O trace (performance trace class 4) or logging trace (performance class 5) is needed as well.

**Tip:** Note that the trace request above does not gather information about DB2 latch contentions, only *page* latch contentions. DB2 latch contention and DB2 page latch contention are not related.

### Other IFCIDs to trace

To understand a concurrency problem, you first need to understand which types of locks are involved, as well as the resources involved and the sequence in which the locks are acquired. However, this does not necessarily indicate why the lock are occurring, that is, it does not indicate which SQL statements are triggering the locks to be acquired. In addition to discovering all the locking information, you usually want to discover which SQL statements are issuing the lock requests that are causing the concurrency problem.

It is also important to understand when locks are released by an application; this can be when a cursor moves off a page, but also at commit time. So we also need to know when transactions commit, as that will release all the update locks, allowing other, suspended, transactions to continue. There are many trace records available for the different types of SQL statements and the different phases and types of commit processing. The simplest way to capture them all is to activate a few performance trace classes instead of listing all the individual IFCIDs.

You can use the following START TRACE command to gather SQL and commit information:

```
-START TRACE(P) CLASS(1,2,3) TDATA(CPU COR DIST) DEST(SMF/GTF)
```

You can limit the amount of trace data by specifying additional filter keywords, such as AUTHID, PLANNAME, or some of the new filters that were introduced in DB2 9, such as PKGCOL, PKGPROG, APPNAME, WRKSTN, CONNID, and CORRID.

When you use OMEGAMON PE batch reporting, you can only use the RECTRACE command to view all the work a transaction has done in DB2. As RECTRACE has a tendency to produce an enormous amount of output, you should always apply as much filtering as possible, using INCLUDE and EXCLUDE, as well as using FROM and TO time stamps to limit the scope of the trace.

## 9.1.5 Using online monitor

Online real-time monitors are usually not very helpful when analyzing locking or serialization problems, especially when dealing with lock suspensions that do not result in a deadlock or timeout. These problems typically only last for a very short time. Therefore, by the time you have selected the suspended thread in your real-time monitor screen, it has already resolved and the relevant information is no longer available.

## 9.2 Analysis of a simple deadlock scenario

This section illustrates a way to analyze deadlock problems. It is applicable to deadlocks and other types of concurrency problems, such as timeouts or long suspension times.

## 9.2.1 Analyzing the joblogs and syslog

When we look at the joblog of the runs (Example 9-22) of our stored procedures workload, we see that there are some performance problems. In this case, the joblog shows that some of the stored procedures, EMPDEL and DPTDEL, received an SQLCODE -913 (deadlock or timeout).

*Example 9-22 Joblog with SQLCODE -913*

---

```
...
GLWR141I: Execution of Stored Procedures; Time: 13:30:08
  1 DPTUPD   13:30:08 ; Objno:    3679 ; Tran:   45182 ; SQLrc: 0
  2 EMPUPD   13:30:08 ; Objno:      0 ; Tran:   45184 ; SQLrc: 0
  3 PRJUPD   13:30:08 ; Objno:   1004 ; Tran:   45186 ; SQLrc: 0
  4 DPTUPD   13:30:08 ; Objno:   4392 ; Tran:   45187 ; SQLrc: 0
  5 PRJADD   13:30:10 ; Objno:   7737 ; Tran:   45188 ; SQLrc: 0
  6 PRJADD   13:30:13 ; Objno:   7739 ; Tran:   45197 ; SQLrc: 0
  7 DPTUPD   13:30:13 ; Objno:   1219 ; Tran:   45200 ; SQLrc: 0
  8 DPTUPR   13:30:13 ; Objno:   2267 ; Tran:   45201 ; SQLrc: 0
  9 EMPUPD   13:30:13 ; Objno:      0 ; Tran:   45202 ; SQLrc: 0
 10 DPTMGR   13:30:13 ; Objno:   5111 ; Tran:   45203 ; SQLrc: 0
 11 DPTUPR   13:30:13 ; Objno:   3422 ; Tran:   45204 ; SQLrc: 0
 12 EMPADD   13:30:13 ; Objno:  30913 ; Tran:   45205 ; SQLrc: 0
 13 EMPQR2   13:30:13 ; Objno:    86 ; Tran:   45206 ; SQLrc: 0
 14 EMPADD   13:30:13 ; Objno:  30914 ; Tran:   45207 ; SQLrc: 0
 15 EMPDEL   13:30:15 ; Objno:      0 ; Tran:   45208 ; SQLrc: -913
 16 EMPDEL   13:30:17 ; Objno:      0 ; Tran:   45209 ; SQLrc: -913
 17 EMPQR2   13:30:17 ; Objno:   2820 ; Tran:   45210 ; SQLrc: 0
 18 DPTBAL   13:30:17 ; Objno:   2627 ; Tran:   45211 ; SQLrc: 0
 19 DPTMGR   13:30:17 ; Objno:      0 ; Tran:   45212 ; SQLrc: 0
 20 EMPFND   13:30:17 ; Objno:    87 ; Tran:   45213 ; SQLrc: 0
...
106 EMPQR2   13:30:45 ; Objno:    707 ; Tran:   45369 ; SQLrc: 0
107 DPTDEL   13:30:47 ; Objno:      0 ; Tran:   45370 ; SQLrc: -913
108 EMPUPD   13:30:47 ; Objno:      0 ; Tran:   45371 ; SQLrc: 0
...
```

---

To confirm this situation, we look at the syslog (Example 9-23) where we see either DSNT375I (deadlock) or DSNT376I (timeout) messages, accompanied by DSNT501I messages.

*Example 9-23 Messages in the SYSLOG*

---

```

2009219 13:30:15.61 STC08083 00000010 DSNT375I -DB9A PLAN=DSNREXX WITH 607
                                607 00000010 CORRELATION-ID=BARTSQR2
                                607 00000010 CONNECTION-ID=DB2CALL
                                607 00000010 LUW-ID=USIBMSC.SCPDB9A.C49A0E07D844=14469
                                607 00000010 THREAD-INFO=BART:*:*:*
                                607 00000010 IS DEADLOCKED WITH PLAN=DSNREXX WITH
                                607 00000010 CORRELATION-ID=BARTSQR1
                                607 00000010 CONNECTION-ID=DB2CALL
                                607 00000010 LUW-ID=USIBMSC.SCPDB9A.C49A0E034547=14468
                                607 00000010 THREAD-INFO=BART:*:*:*
                                607 00000010 ON MEMBER DB9A
2009219 13:30:15.62 STC08083 00000010 DSNT501I -DB9A DSNILMCL RESOURCE UNAVAILABLE 608
                                608 00000010 CORRELATION-ID=BARTSQR2
                                608 00000010 CONNECTION-ID=DB2CALL
                                608 00000010 LUW-ID=USIBMSC.SCPDB9A.C49A0E07D844=14469
                                608 00000010 REASON 00C90088
                                608 00000010 TYPE 00000302
                                608 00000010 NAME GLWBSQLP.GLWSPRJ .X'000243'

```

---

In this example, the deadlock produced -913 SQLCODEs.

## 9.2.2 Analyzing the deadlock record

As the system is running with the recommended statistics traces (which includes statistics trace class 3), we also have IFCID 172 (deadlock) trace records. We discover those records by running an OMEGAMON PE lockout trace using the following command:

```

DB2PM
GLOBAL
  TIMEZONE (+4)
LOCKING
  TRACE
    LEVEL(LOCKOUT)
EXEC

```

An excerpt of the report is shown in Example 9-24. The lockout trace contains several, similar deadlocks. The same resources are involved, but different pages are triggering different deadlocks.

*Example 9-24 LOCKOUT trace*

---

```

LOCATION: DB9A                                OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)                                PAGE: 1-1
      GROUP: N/P                                LOCKING TRACE - LOCKOUT                                REQUESTED FROM: NOT SPECIFIED
      MEMBER: N/P                                ACTUAL FROM: 08/07/09 13:30:38.62                                TO: NOT SPECIFIED
      SUBSYSTEM: DB9A                                PAGE DATE: 08/07/09
DB2 VERSION: V9                                SCOPE: MEMBER
PRIMAUTH CORRNAME CONNTYPE                                --- L O C K   R E S O U R C E ---
ORIGAUTH CORRNMBR INSTANCE                                EVENT TIMESTAMP                                --- L O C K   R E S O U R C E ---
PLANNAME CONNECT                                RELATED TIMESTAMP EVENT                                TYPE NAME                                EVENT SPECIFIC DATA
-----
BART    BARTSQR2 DB2CALL    13:30:38.62062331 DEADLOCK                                COUNTER = 928K    WAITERS = 2
BART    'BLANK'  C49A0E07D844 N/P                                TSTAMP  =08/07/09 13:30:38.61
DSNREXX DB2CALL                                HASH    =X'009CCA43'
                                ----- BLOCKER is HOLDER -----
                                LUW=USIBMSC.SCPDB9A.C49A0E034547
                                MEMBER  =N/P    CONNECT =DB2CALL
                                PLANNAME=DSNREXX  CORRID  =BARTSQR1
                                DATAPAGE DB  =GLWBSQLP
                                OB    =31
                                PAGE=X'000243'

```

---

```

DURATION=COMMIT    PRIMAUTH=BART
STATE    =X
PROGNAME=EMPSEL
COLLID   =GLWBSQLP
LOCATION=N/P
CONTOKEN=X'17BFB71C0BD53AB0'
----- WAITER -----*VICTIM*-
LUW=USIBMSC.SCPDB9A.C49A0E07D844
MEMBER   =N/P      CONNECT =DB2CALL
PLANNAME=DSNREXX  CORRID  =BARTSQR2
DURATION=MANUAL   PRIMAUTH=BART
REQUEST  =LOCK    WORTH   = 17
STATE    =U
PROGNAME=EMPDEL
COLLID   =GLWBSQLP
LOCATION=N/P
CONTOKEN=X'17BFD3BB057A566F'

DATAPAGE DB  =GLWBSQLP      HASH    =X'00C248EF'
OB  =GLWSEMP              ----- BLOCKER is HOLDER --*VICTIM*-
PAGE=X'3000EF'           LUW=USIBMSC.SCPDB9A.C49A0E07D844
                        MEMBER   =N/P      CONNECT =DB2CALL
                        PLANNAME=DSNREXX  CORRID  =BARTSQR2
                        DURATION=COMMIT   PRIMAUTH=BART
                        STATE    =X
                        PROGNAME=EMPDEL
                        COLLID   =GLWBSQLP
                        LOCATION=N/P
                        CONTOKEN=X'17BFD3BB057A566F'
                        ----- WAITER -----
                        LUW=USIBMSC.SCPDB9A.C49A0E034547
                        MEMBER   =N/P      CONNECT =DB2CALL
                        PLANNAME=DSNREXX  CORRID  =BARTSQR1
                        DURATION=MANUAL   PRIMAUTH=BART
                        REQUEST  =LOCK    WORTH   = 18
                        STATE    =S
                        PROGNAME=EMPSEL
                        COLLID   =GLWBSQLP
                        LOCATION=N/P
                        CONTOKEN=X'17BFB71C0BD53AB0'

```

...

In Example 9-24 on page 282, the following transactions and resources are involved:

- ▶ BARTSQR1(stored procedure EMPSEL) is holding an X lock on datapage GLWBSQLP.31.x'000243', while BARTSQR2 (stored procedure EMPDEL) is waiting for a U lock on that resource.
- ▶ BARTSQR2 is holding an X lock on data page GLWBSQLP.GLWSEMP.x'3000EF' (this is page x'EF' in partition for of the GLWSEMP partitioned table space), while BARTSQR1 is waiting for an S lock on the same resource.

This is a classic example of a deadlock, where BARTSQR2 is the victim.

OBID 31 in the GLWBSQLP database is part of the GLWTPRJ table. Figure 9-1 shows the result of querying the catalog using the DB2 Admin Tool.

```

DB2 Admin ----- DB9A Interpretation of an Object in SYSTABLES ----- 02:02
Option ==>

Details for table (label) : GLWBSQLP.GLWTPRJ

Table schema      : GLWBSQLP      Table name        : GLWTPRJ
Created by        : BART           Created           : 2009-08-06-20.38.50.385435
Table space name  : GLWSPRJ        Database name      : GLWBSQLP
Object ID for table: 31            DB ID for database : 754
...

```

Figure 9-1 List OBID 31

**Note:** Note that the time stamp of the deadlock record is 13:30:38, while the time stamp of the DSNT375I and the time stamp in the joblog of the SQLCODE -913 is 13:30:15, which is a 23 second difference. This is because OMEGAMON PE currently does not allow you to use a TIMEZONE that includes seconds (only hours and minutes).

However, this System z machine is connected to an external timer facility that is configured to use leap seconds (the delta between UTC (Coordinated Universal Time) and UT1 (mean solar time - observed Earth rotation)).

The DB2 trace records use an STCK time stamp, which is not adjusted for leap seconds. The console messages and job log messages use the local time (including the leap seconds).

The external timer is currently set to use 23 leap seconds, which is why there is a 23 second difference between the time in the OMEGAMON PE reports and the console messages. (The number of leap seconds should be 24, but for some reason the most recent leap second has not been added yet.)

### 9.2.3 Checking the DB2 accounting data

To get a better understanding of how much impact these deadlocks have on the performance of our workload (BARTSQR1 and BARTSQR2 jobs), we generate an OMEGAMON PE accounting report using the following statements:

```
DB2PM
GLOBAL
  TIMEZONE (+4)
  INCLUDE(DB2ID(DB9A)
          PLANNAME(DSNREXX))
ACCOUNTING
  REPORT
    LAYOUT (LONG)
    ORDER (PLANNAME)
EXEC
```

An extract of the report is shown in Example 9-25 on page 285, where:

- ▶ The workload is almost exclusively made up of stored procedures, so it is no surprise that the vast majority of the elapsed and CPU time is for stored procedures.
- ▶ As expected, there are two occurrences, BARTSQR1 and BARTSQR2.
- ▶ Out of the 1:02 minutes that each job ran, on average they spent 18.10 seconds waiting for a lock or a latch (IRLM and DB2).
- ▶ Note that there were six deadlocks during this run.

The high lock/latch wait time and six deadlocks in one minute are clearly unacceptable.

Example 9-25 OMEGAMON PE accounting report by plan name

LOCATION: DB9A	OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)	PAGE: 1-1
GROUP: N/P	ACCOUNTING REPORT - LONG	REQUESTED FROM: NOT SPECIFIED
MEMBER: N/P		TO: NOT SPECIFIED
SUBSYSTEM: DB9A	ORDER: PLANNAME	INTERVAL FROM: 08/07/09 13:31:28.70
DB2 VERSION: V9	SCOPE: MEMBER	TO: 08/07/09 13:31:33.95

PLANNAME: DSNREXX

ELAPSED TIME DISTRIBUTION

```

APPL  => 2%
DB2   =====> 67%
SUSP  =====> 31%
  
```

CLASS 2 TIME DISTRIBUTION

```

CPU   =====> 59%
SECPU =====> 9%
NOTACC =====> 32%
SUSP  =====> 32%
  
```

AVERAGE	APPL (CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	AVERAGE TIME	AV.EVENT	HIGHLIGHTS
<b>ELAPSED TIME</b>	<b>1:03.28040</b>	<b>1:02.21103</b>	N/P	<b>LOCK/LATCH(DB2+IRLM)</b>	<b>18.102153</b>	<b>1508.00</b>	<b>#OCCURRENCES : 2</b>
NONNESTED	2.630104	2.366610	N/A	SYNCHRON. I/O	0.757503	556.50	#ALLIEDS : 2
<b>STORED PROC</b>	<b>1:00.61762</b>	<b>59.811741</b>	N/A	DATABASE I/O	0.208060	280.00	#ALLIEDS DISTRIB: 0
UDF	0.000000	0.000000	N/A	LOG WRITE I/O	0.549443	276.50	#DBATS : 0
TRIGGER	0.032677	0.032677	N/A	OTHER READ I/O	0.320592	368.50	#DBATS DISTRIB. : 0
				OTHER WRTE I/O	0.000000	0.00	#NO PROGRAM DATA: 0
CP CPU TIME	37.454643	36.919796	N/P	SER.TASK SWTCH	0.418567	193.50	#NORMAL TERMINAT: 2
AGENT	37.454643	36.919796	N/A	UPDATE COMMIT	0.403028	182.00	#DDFRRSAF ROLLUP: 0
NONNESTED	0.477001	0.344800	N/P	OPEN/CLOSE	0.000000	0.00	#ABNORMAL TERMIN: 0
STORED PROC	36.956643	36.553997	N/A	SYSLGRNG REC	0.007485	5.50	#CP/X PARALLEL. : 0
UDF	0.000000	0.000000	N/A	EXT/DEL/DEF	0.000000	0.00	#IO PARALLELISM : 0
TRIGGER	0.020999	0.020999	N/A	OTHER SERVICE	0.008053	6.00	#INCREMENT. BIND: 132
PAR.TASKS	0.000000	0.000000	N/A	ARC.LOG(QUIES)	0.000000	0.00	#COMMITTS : 362
				LOG READ	0.000000	0.00	#ROLLBACKS : 6
SECP CPU	0.000000	N/A	N/A	DRAIN LOCK	0.000000	0.00	#SVPT REQUESTS : 507
				CLAIM RELEASE	0.000000	0.00	#SVPT RELEASE : 507
SE CPU TIME	0.000000	0.000000	N/A	PAGE LATCH	0.000000	0.00	#SVPT ROLLBACK : 507
NONNESTED	0.000000	0.000000	N/A	NOTIFY MSGS	0.000000	0.00	MAX SQL CASC LVL: 2
STORED PROC	0.000000	0.000000	N/A	GLOBAL CONTENTION	0.000000	0.00	UPDATE/COMMIT : 10.68
UDF	0.000000	0.000000	N/A	COMMIT PHI WRITE I/O	0.000000	0.00	SYNCH I/O AVG. : 0.001361
TRIGGER	0.000000	0.000000	N/A	ASYNCH CF REQUESTS	0.000000	0.00	
				TCPIP LOB	0.000000	0.00	
PAR.TASKS	0.000000	0.000000	N/A	TOTAL CLASS 3	19.598816	2626.50	
SUSPEND TIME	0.338259	19.598816	N/A				
AGENT	N/A	19.598816	N/A				
PAR.TASKS	N/A	0.000000	N/A				
STORED PROC	0.338259	N/A	N/A				
UDF	0.000000	N/A	N/A				
NOT ACCOUNT.	N/A	5.692417	N/A				
DB2 ENT/EXIT	N/A	1192.00	N/A				
EN/EX-STPROC	N/A	229243.00	N/A				
EN/EX-UDF	N/A	0.00	N/A				
DCAPT.DESCR.	N/A	N/A	N/P				
LOG EXTRACT.	N/A	N/A	N/P				

...

SQL DML	AVERAGE	TOTAL	SQL DCL	TOTAL	SQL DDL	CREATE	DROP	ALTER	LOCKING	AVERAGE	TOTAL
SELECT	5637.00	11274	LOCK TABLE	0	TABLE	0	33	0	TIMEOUTS	0.00	0
INSERT	1625.00	3250	GRANT	0	CRT TTABLE	0	N/A	N/A	<b>DEADLOCKS</b>	<b>3.00</b>	<b>6</b>
UPDATE	335.00	670	REVOKE	0	DCL TTABLE	33	N/A	N/A	ESCAL.(SHARED)	0.00	0
MERGE	0.00	0	SET CURR.SQLID	0	AUX TABLE	0	N/A	N/A	ESCAL.(EXCLUS)	0.00	0
DELETE	6.00	12	SET HOST VAR.	111158	INDEX	0	0	0	MAX PG/ROW LOCKS HELD	32.50	33
			SET CUR.DEGREE	0	TABLESPACE	0	0	0	LOCK REQUEST	7454.00	14908
DESCRIBE	40.00	80	SET RULES	0	DATABASE	0	0	0	UNLOCK REQUEST	851.50	1703
DESC.TBL	0.00	0	SET CURR.PATH	0	STOGROUP	0	0	0	QUERY REQUEST	0.00	0
PREPARE	56.00	112	SET CURR.PREC.	0	SYNONYM	0	0	N/A	CHANGE REQUEST	563.50	1127
OPEN	113.50	227	CONNECT TYPE 1	0	VIEW	0	0	0	OTHER REQUEST	0.00	0
FETCH	49437.50	98875	CONNECT TYPE 2	0	ALIAS	0	0	N/A	TOTAL SUSPENSIONS	22.00	44
CLOSE	97.00	194	SET CONNECTION	0	PACKAGE	N/A	0	N/A	LOCK SUSPENSIONS	20.50	41
			RELEASE	0	PROCEDURE	0	0	0	IRLM LATCH SUSPENS.	1.50	3
DML-ALL	57347.00	114694	CALL	3028	FUNCTION	0	0	0	OTHER SUSPENS.	0.00	0
			ASSOC LOCATORS	0	TRIGGER	0	0	N/A			
			ALLOC CURSOR	0	DIST TYPE	0	0	N/A			
			HOLD LOCATOR	0	SEQUENCE	0	0	0			
			FREE LOCATOR	0	TRUST. CTX	0	0	0			
			DCL-ALL	114186	ROLE	0	0	N/A			
					JAR	N/A	N/A	0			
					TOTAL	33	33	0			

If you have DB2 package level accounting (classes 7 and 8) active, you can also get information at the package level that can help you identify the packages that are suffering from a high lock/latch contention. If accounting class 10 is also active, the package level trace record contains additional information about the locking activity that the package performed.

Example 9-26 shows the package level accounting information of one of the packages that has a high lock/latch wait time. The package name is EMPDEL, and is invoked as a stored procedure with the same name.

*Example 9-26 EMPDEL package level accounting info class 7 and 8*

EMPDEL	VALUE	EMPDEL	TIMES	EMPDEL	AVERAGE TIME	AVG.EV	TIME/EVENT
-----	-----	-----	-----	-----	-----	-----	-----
TYPE	PACKAGE	ELAP-CL7 TIME-AVG	0.356771	<b>LOCK/LATCH</b>	<b>0.279303</b>	<b>0.17</b>	<b>1.675819</b>
		CP CPU TIME	0.044015	SYNCHRONOUS I/O	0.007594	8.83	0.000860
LOCATION	DB9A	AGENT	0.044015	OTHER READ I/O	0.015602	24.75	0.000630
COLLECTION ID	GLWBSQLP	PAR.TASKS	0.000000	OTHER WRITE I/O	0.000000	0.00	N/C
PROGRAM NAME	EMPDEL	SE CPU TIME	0.000000	SERV.TASK SWITCH	0.000447	0.33	0.001340
		SUSPENSION-CL8	0.302946	ARCH.LOG(QUIESCE)	0.000000	0.00	N/C
ACTIVITY TYPE	STORED PROC	AGENT	0.302946	ARCHIVE LOG READ	0.000000	0.00	N/C
ACTIVITY NAME	EMPDEL	PAR.TASKS	0.000000	DRAIN LOCK	0.000000	0.00	N/C
SCHEMA NAME	GLWBSQLP	NOT ACCOUNTED	0.009810	CLAIM RELEASE	0.000000	0.00	N/C
OCCURRENCES	2	AVG.DB2 ENTRY/EXIT	4.00	PAGE LATCH	0.000000	0.00	N/C
NBR OF ALLOCATIONS	24	DB2 ENTRY/EXIT	96	NOTIFY MESSAGES	0.000000	0.00	N/C
SQL STMT - AVERAGE	1.13			GLOBAL CONTENTION	0.000000	0.00	N/C
SQL STMT - TOTAL	27	CP CPU SU	1248.63	TCP/IP LOB	0.000000	0.00	N/C
SUCC AUTH CHECK	0	AGENT	1248.63	TOTAL CL8 SUSPENS.	0.302946	34.08	0.008888
		PAR.TASKS	0.00				
		SE CPU SU	0.00				

Example 9-27 shows the additional package information that is gathered when accounting class 10 is active. Note that this package was a victim of a deadlock four times. This is consistent with the information from the IFCID 172 records that show four deadlocks where EMPDEL is involved.

*Example 9-27 EMPDEL package level accounting info class 10*

EMPDEL	AVERAGE	TOTAL
-----	-----	-----
SELECT	0.13	3
INSERT	0.13	3
UPDATE	0.00	0
DELETE	0.29	7
DESCRIBE	0.00	0
PREPARE	0.00	0
OPEN	0.00	0
FETCH	0.00	0
CLOSE	0.00	0
LOCK TABLE	0.00	0
CALL	0.58	14
EMPDEL	AVERAGE	TOTAL
-----	-----	-----
BPOOL HIT RATIO (%)	18.09	N/A
GETPAGES	1234.54	29629
BUFFER UPDATES	9.33	224
SYNCHRONOUS WRITE	0.00	0
SYNCHRONOUS READ	8.33	200
SEQ. PREFETCH REQS	34.46	827



LIST PREFETCH REQS	0.00	0
DYN. PREFETCH REQS	0.00	0
PAGES READ ASYNCHR.	1002.92	24070

EMPDEL	AVERAGE	TOTAL
-----	-----	-----
TIMEOUTS	0.00	0
<b>DEADLOCKS</b>	<b>0.17</b>	<b>4</b>
ESCAL.(SHARED)	0.00	0
ESCAL.(EXCLUS)	0.00	0
MAX PG/ROW LOCKS HELD	0.00	0
LOCK REQUEST	16.58	398
UNLOCK REQUEST	12.54	301
QUERY REQUEST	0.00	0
CHANGE REQUEST	0.25	6
OTHER REQUEST	0.00	0
TOTAL SUSPENSIONS	0.17	4
<b>LOCK SUSPENSIONS</b>	<b>0.17</b>	<b>4</b>
IRLM LATCH SUSPENS.	0.00	0
OTHER SUSPENS.	0.00	0

---

## 9.2.4 Zooming in using the record trace

The deadlock trace record reveals the involved resources and transactions and how their lock requests became chronically deadlocked. We must discover the sequence in which the locks were requested and which SQL statements triggered these lock requests.

The deadlock record provides the time the deadlock occurred and the instance numbers of the transactions involved. The instance number is part of the LUW-ID that uniquely identifies a transaction, and it can be used as a filter criteria for OMEGAMON PE.

### History of the deadlock victim

Unfortunately, there is no easy way to construct a full picture of a locking problem. Unless you can identify the problem by looking at the program names and resources involved, you almost always have to go down to the detailed trace level to get to all the information you need. We used the OMEGAMON PE RECTRACE to obtain the information we need by using the following statements:

```
GLOBAL
  TIMEZONE(+4)
  FROM(,13:30:35)
  TO(,13:30:40)
  INCLUDE (
    DB2ID(DB9A)
    INSTANCE(C49A0E07D844)
  )
RECTRACE
  TRACE
  LEVEL(LONG)
EXEC
```

The deadlock occurred at 13:30:38.62062331, so we select a time frame around the deadlock time. This often results in a fairly large report, so we only discuss some excerpts of it in this publication.

### ***The lock request that resulted in a deadlock***

To verify that we are looking at the right data, we perform a search for the 'DEADLOCK' keyword in the output from the RECTRACE to make sure that the IFCID 172 is in the trace. As this record trace is for the deadlock victim, the trace output should contain a deadlock record. Example 9-28 shows this information. Note the following items:

- ▶ C49A0E07D844 is our deadlock victim.
- ▶ At 13:30:36.71860151, there was a request for a U lock on GLWBSQLP.31. X'00024300' was suspended, and at 13:30:38.61881981, the lock request was resumed with reason DEADLOCK.
- ▶ The IFCID 172 information was shortened, as we have already looked at this information in the LOCKOUT trace.
- ▶ At 13:30:38.62069458, the lock request ends with RC=8 (not granted) and reason code x'20', which indicates that the lock was not granted because a deadlock or timeout occurred.

**Example 9-28** Lock request triggering the deadlock

LOCATION: DB9A		OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)					PAGE: 1-720	
GROUP: N/P		RECORD TRACE - LONG					REQUESTED FROM: ALL 13:30:35.00	
MEMBER: N/P							TO: DATES 13:30:40.00	
SUBSYSTEM: DB9A							ACTUAL FROM: 08/07/09 13:30:35.00	
DB2 VERSION: V9							PAGE DATE: 08/07/09	
PRIMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME	TRANSACTION			
ORIGAUTH	CORRNAME	CONNTYPE	RECORD TIME	DESTNO ACE	IFC	DESCRIPTION	DATA	
PLANNAME	CORRNMBR		TCB CPU TIME		ID			
-----								
....								
BART	DB2CALL	C49A0E07D844	'BLANK'	'BLANK'			'BLANK'	
BART	BARTSQR2	DB2CALL	13:30:36.71857853	922836	1	21	LOCK DETAIL	NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
DSNREXX	'BLANK'		9.91268987					
-----								
LOCK RES TYPE: N/P			NAME: N/P					
IRLM FUNC CODE : UNLOCK (TOKEN)			RETURN TOKEN: X'00000000'			REQUEST TOKEN : X'7F6ACF30'		
LOCK STATE : UPDATE			DB2 TOKEN : X'00A4B00024C0A428'			IRLM RETURN CODE : 0		
LOCK ATTRIBUTES: NMODIFY NOFORCE			PROP TO XES : NO			ASYN TO XES : NO		
LOCK DURATION : MANUAL + 1			REQUEST TYPE:			IRLM RETURN SUBCODE: B'0000000000000000'		
PARENT TOKEN : X'00000000'			GLOBAL/LOCAL: GLOBAL			OWNER : 'BLANK'		
CACHED STATE : N/A						LOCK HASH VALUE : X'009C8A42'		
QW0021CL: X'00'		QW0021U : X'00A900E520BCE600'		QW0021CT: X'7C28A500'		QW0021FL: B'00110000'		
QW0021F3: B'00000000'		QW0021O : X'00A900E520BCE558'		QW0021IR: X'0000'		QW0021F2: B'00000000'		
-----								
13:30:36.71860151				922837	1	44	LOCK SUSPEND	'BLANK'
9.91271138				NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1				
-----								
LOCK RES TYPE: DATA PAGE LOCK		DBID: GLWBSQLP		OBID: 31		RESOURCE ID: X'00024300'		
IRLM FUNC CODE: LOCK (NAME)		STATE: UPDATE		DURATION: MANUAL		REASON SUSP: LC		
REQ TOKEN: X'00000000'		LOCK ATTRIBUTES: L-LOCK GLOBAL NOMODIFY NOFORCE		PROP TO XES: NO		ASYN TO XES: NO		
PARENT TOKEN: X'7F69BCA0'		LOCK HASH VALUE: X'009CCA43'						
QW0044CL: X'00'		QW0044FL: X'30'						
-----								
13:30:38.61881981				922842	1	45	LOCK	<-- 'BLANK'
9.91273428				RESUME				
				NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1				
				REASON FOR RESUME : DEADLOCK				
				REASON FOR SUSPEND : X'20'				
				IRLM LATCH CONTENTION : NO				
				IRLM QUEUED REQUEST : NO				
				LOCAL RESOURCE CONTENTION : YES				
				RETAINED LOCK CONTENTION : NO				
				GLOBAL RESOURCE CONTENTION : NO				
				INTER-SYSTEM MESSAGE SENDING : NO				
				GLOBAL CONTENTION EXTENT : X'00'				
				XES GLOBAL CONTENTION : NO				
				IRLM GLOBAL CONTENTION : NO				
				FALSE CONTENTION : NO				
QW0045W4		NO	QW0045W6	NO	QW0045W8	NO		
QW0045X1		NO	QW0045X2	NO	QW0045X5	NO		
QW0045X6		NO	QW0045X7	NO	QW0045X8	NO		

```

BART DB2CALL C49A0E07D844 'BLANK' 'BLANK' 'BLANK'
BART BARTSRQ2 DB2CALL 13:30:38.62062331 922843 1 172 DEADLOCK DATA NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
DSNREXX 'BLANK' 9.91284116
-----
DEADLOCK HEADER
INTERVAL COUNT: 928463 WAITERS INVOLVED: 2 TIME DETECTED: 08/07/09 17:30:38.618762
-----
UNIT OF WORK
R E S O U R C E
....
13:30:38.62069458 922844 1 21 LOCK DETAIL 'BLANK'
9.91290456 NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
-----
LOCK RES TYPE: DATA PAGE LOCK DBID: GLWBSQLP OBID: 31 RESOURCE ID: X'00024300'
IRLM FUNC CODE : LOCK (NAME) RETURN TOKEN: X'00000000' REQUEST TOKEN : X'00000000'
LOCK STATE : UPDATE DB2 TOKEN : X'00A4B00024C0A428' IRLM RETURN CODE : 8
LOCK ATTRIBUTES: NMODIFY NOFORCE PROP TO XES : NO ASYN TO XES : NO
LOCK DURATION : MANUAL REQUEST TYPE: IRLM RETURN SUBCODE: B'0010000000000000'
PARENT TOKEN : X'7F69BCA0' GLOBAL/LOCAL: GLOBAL OWNER : 'BLANK'
CACHED STATE : N/A LOCK HASH VALUE : X'009CCA43'
QW0021CL: X'00' QW0021U : X'00A900E520BCE600' QW0021CT: X'7C28A500' QW0021FL: B'00110000'
QW0021F3: B'00000000' QW00210 : X'00A900E520BCE558' QW0021IR: X'0000' QW0021F2: B'00000000'
-----

```

**Tip:** IFCID 21 (detailed lock request) is only written when the request has completed, as it contains the return code and reason code of the request.

### ***The SQL statement that triggered the deadlock***

After we discover that the lock request resulted in a deadlock, we check to see which SQL statement was being executed at the time. If the trace has the recommended performance trace class(1,2,3) information, it will also contain the SQL statement begin and end trace records. We go back in time starting from the deadlock record until we find the beginning of the SQL statement that triggered it. (Sometimes is easier to search for the end SQL trace record of the previous SQL statement (IFCID 58) and then go a bit forward again.)

Example 9-29 shows that the triggering SQL statement is a DELETE statement.

*Example 9-29 SQL statement triggering the deadlock*

```

13:30:36.71210628 922774 1 61 DELETE --> 'BLANK'
9.90665277 START NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
LOCATION NAME : DB9A
PKG COLLECTION ID : GLWBSQLP
PROGRAM NAME : EMPDEL
CONSISTENCY TOKEN : X'17BFD3BB057A566F'
STATEMENT NUMBER : 374
STATEMENT TYPE : DELETE TYPE- NON CURSOR
CURSOR NAME : N/P
QUERY COMMAND ID : N/P
QUERY INSTANCE ID : N/P
ISOLATION : CS
REOPTIMIZATION : NO

```

The SQL statement is statement 374 in the EMPDEL package. This is a static SQL package, so we can use the DB2 Admin tool to look at the SQL statement text (see Figure 9-2). The failing SQL statement is:

```
DELETE FROM GLWTEMP WHERE EMP_NO = :H :H
```

This statement removes an employee from the GLWTEMP table.

```
DB2 Admin ----- Extracted SQL ----- Columns 00001 00072
Command ==>                                     Scroll ==> CSR
***** ***** Top of Data *****
=NOTE= -- SQL statements in PACKAGE : GLWBSQLP.EMPDEL.()
=NOTE= -- SQL in stmt: 221
000001 SELECT 0 INTO :H FROM SYSIBM.SYSDUMMY1 WITH UR
=NOTE= -- SQL in stmt: 259
000002 CALL SPLANO (:H :H, :H :H)
=NOTE= -- SQL in stmt: 312
000003 CALL EMPSEL (:H :H, :H :H, :H :H, :H :H, :H :H)
=NOTE= -- SQL in stmt: 374
000004 DELETE FROM GLWTEMP WHERE EMP_NO = :H :H
=NOTE= -- SQL in stmt: 408
000005 SELECT WL_BUILDRI INTO :H :H FROM GLWTVRN
=NOTE= -- SQL in stmt: 432
000006 UPDATE GLWTPRJ SET RESP_EMP_NO = NULL WHERE RESP_EMP_NO = :H :H
=NOTE= -- SQL in stmt: 448
000007 UPDATE GLWTDPT SET MGR_NO = NULL WHERE MGR_NO = :H :H
=NOTE= -- SQL in stmt: 463
000008 DELETE FROM GLWTEPA WHERE EMP_NO = :H :H
=NOTE= -- SQL in stmt: 494
000009 INSERT INTO GLWTSPL (TRAN_NO, CALL_SQLID, CALL_PROC, CALL_RC,
000010 CALL_OBJ_NO) VALUES (:H :H, USER, 'EMPDEL', 0, :H :H)
***** ***** Bottom of Data *****
```

Figure 9-2 SQL statements of EMPDEL package

The U lock request that resulted in the deadlock was for GLWBSQLP.31. X'00024300', where OBID 31 is the *GLWTPRJ* table (see Figure 9-1 on page 283), but the DELETE statement is on the *GLWTEMP* table. Before we look at this situation in more detail, let us discover the other involved SQL statements.

When the DELETE statement starts, DB2 first locks the partition in IX mode, and then acquires an X lock on DBID: GLWBSQLP,OBID: GLWSEMP, RESOURCE ID: X'3000EF00', as shown in Example 9-30. Note that the resource ID includes the x'00' at the end. It is not part of the page number; it is the ID map entry. As we are using page level locking, this will always be zero.

**Example 9-30** Acquiring the X-lock on page x'3000EF'

LOCATION: DB9A		OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)					PAGE: 1-702	
GROUP: N/P		RECORD TRACE - LONG					REQUESTED FROM: ALL 13:30:35.00	
MEMBER: N/P							TO: DATES 13:30:40.00	
SUBSYSTEM: DB9A							ACTUAL FROM: 08/07/09 13:30:35.00	
DB2 VERSION: V9							PAGE DATE: 08/07/09	
PRIMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME		TRANSACT		
ORIGAUTH	CORRNAME	CONNTYPE	RECORD TIME	DESTNO ACE	IFC	DESCRIPTION	DATA	
PLANNAME	CORRNMBR		TCB CPU TIME		ID			
-----								
			13:30:36.71210628	922774	1	61	DELETE	--> 'BLANK'
			9.90665277	START				
			NETWORKID: USIBMSC					LUNAME: SCPDB9A LUWSEQ: 1
			LOCATION NAME					: DB9A
			PKG COLLECTION ID					: GLWBSQLP
			PROGRAM NAME					: EMPDEL
			CONSISTENCY TOKEN					: X'17BFD3BB057A566F'
			STATEMENT NUMBER					: 374
			STATEMENT TYPE					: DELETE TYPE- NON CURSOR
			CURSOR NAME					: N/P
			QUERY COMMAND ID					: N/P
			QUERY INSTANCE ID					: N/P
			ISOLATION					: CS
			REOPTIMIZATION					: NO
BART	DB2CALL	C49A0E07D844	'BLANK'	'BLANK'				
BART	BARTSR2	DB2CALL	13:30:36.71216069	922775	1	21	LOCK DETAIL	NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
DSNREXX	'BLANK'		9.90670345					
-----								
LOCK RES TYPE: PARTITION LOCK WITH SPL				DBID: GLWBSQLP		OBID: GLWSEMP		
IRLM FUNC CODE : LOCK (NAME)				RETURN TOKEN: X'7F6AC1B0'		REQUEST TOKEN : X'00000000'		
LOCK STATE : INTENT EXCLUSIVE				DB2 TOKEN : X'00A4B00024C0A428'		IRLM RETURN CODE : 0		
LOCK ATTRIBUTES: MODIFY NOFORCE				PROP TO XES : NO		ASYN TO XES : NO		
LOCK DURATION : COMMIT				REQUEST TYPE:		IRLM RETURN SUBCODE: B'0000000000000000'		
PARENT TOKEN : X'00000000'				GLOBAL/LOCAL: GLOBAL		OWNER : 'BLANK'		
LOCKED STATE : N/A						LOCK HASH VALUE : X'00020DF2'		
QW0021CL: X'00'				QW0021U : X'00A900E520BCE600'		QW0021CT: X'7C28A500'		QW0021FL: B'10110010'
QW0021F3: B'00000000'				QW00210 : X'00A900E520BCE558'		QW0021IR: X'0000'		QW0021F2: B'00000000'
-----								
			13:30:36.71216898	922776	1	21	LOCK DETAIL	'BLANK'
			9.90670891	NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1				
-----								
LOCK RES TYPE: DATA PAGE LOCK				DBID: GLWBSQLP		OBID: GLWSEMP RESOURCE ID: X'3000EF00'		
IRLM FUNC CODE : LOCK (NAME)				RETURN TOKEN: X'7F6AD650'		REQUEST TOKEN : X'00000000'		
LOCK STATE : EXCLUSIVE				DB2 TOKEN : X'00A4B00024C0A428'		IRLM RETURN CODE : 0		
LOCK ATTRIBUTES: MODIFY NOFORCE				PROP TO XES : NO		ASYN TO XES : NO		
LOCK DURATION : COMMIT				REQUEST TYPE:		IRLM RETURN SUBCODE: B'0000000000000000'		
PARENT TOKEN : X'7F6AC1B0'				GLOBAL/LOCAL: GLOBAL		OWNER : 'BLANK'		
LOCKED STATE : N/A						LOCK HASH VALUE : X'00C248EF'		
QW0021CL: X'00'				QW0021U : X'00A900E500000000'		QW0021CT: X'7C28A500'		QW0021FL: B'10110010'
QW0021F3: B'00000000'				QW00210 : X'00A900E520BCE558'		QW0021IR: X'0000'		QW0021F2: B'00010000'
-----								

This is the same DELETE statement that first issues the X lock on the GLWSEMP table space (and acquires it, as the IRLM return code is zero), and then later issues an S lock on the GLWTPRJ table, which results in a deadlock.

## History of the other transaction involved in the deadlock

In this case, only one other transaction was involved, BARTSQR1, which has instance number C49A0E034547. We now try to discover what this transaction was doing when the deadlock occurred. We use the following OMEGAMON PE command string to obtain this information:

```
DB2PM
GLOBAL
  TIMEZONE(+4)
  FROM(,13:30:35)
  TO(,13:30:40)
  INCLUDE (
    DB2ID(DB9A)
    INSTANCE(C49A0E034547)
  )
RECTRACE
  TRACE
    LEVEL(LONG)
EXEC
```

We use the instance number and the same time interval that we used for the deadlock victim. An excerpt of the report is shown in Example 9-31.

As both transactions are involved in the deadlock, this transaction will also have been suspended at some point, until DB2 decided to end the deadlock by aborting the other transaction. We search the trace for a time stamp close to the deadlock time stamp of 13:30:38.

Example 9-31 RECTRACE of deadlock survivor

```
....
      13:30:36.72817953 922841 1 44 LOCK SUSPEND 'BLANK'
      43.04724955 NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
-----
LOCK RES TYPE: DATA PAGE LOCK DBID: GLWBSQLP OBID: GLWSEMP RESOURCE ID: X'3000EF00'
IRLM FUNC CODE: LOCK (NAME) STATE: SHARED DURATION: MANUAL REASON SUSP: LC
REQ TOKEN: X'00000000' LOCK ATTRIBUTES: L-LOCK GLOBAL NOMODIFY NOFORCE PROP TO XES: NO ASYN TO XES: NO
PARENT TOKEN: X'7F69C100' LOCK HASH VALUE: X'00C248EF'
QW0044CL: X'00' QW0044FL: X'30'
-----
LOCATION: DB9A OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2) PAGE: 1-247
GROUP: N/P RECORD TRACE - LONG REQUESTED FROM: ALL 13:30:35.00
MEMBER: N/P TO: DATES 13:30:40.00
SUBSYSTEM: DB9A ACTUAL FROM: 08/07/09 13:30:35.09
DB2 VERSION: V9 PAGE DATE: 08/07/09
PRIMAUTH CONNECT INSTANCE END_USER WS_NAME TRANSACT
ORIGAUTH CORRNAME CONNTYPE RECORD TIME DESTNO ACE IFC DESCRIPTION DATA
PLANNAME CORRNMBR TCB CPU TIME ID
-----
BART DB2CALL C49A0E034547 'BLANK' 'BLANK' 'BLANK'
BART BARTSQR1 DB2CALL 13:30:38.62763403 922871 1 45 LOCK <-- NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
DSNREXX 'BLANK' 43.04726098 RESUME REASON FOR RESUME : NORMAL RESUME
REASON FOR SUSPEND : X'20'
IRLM LATCH CONTENTION : NO
IRLM QUEUED REQUEST : NO
LOCAL RESOURCE CONTENTION : YES
RETAINED LOCK CONTENTION : NO
GLOBAL RESOURCE CONTENTION : NO
INTER-SYSTEM MESSAGE SENDING : NO
GLOBAL CONTENTION EXTENT : X'00'
XES GLOBAL CONTENTION : NO
IRLM GLOBAL CONTENTION : NO
FALSE CONTENTION : NO
QW0045W4 NO QW0045W6 NO QW0045W8 NO
QW0045X1 NO QW0045X2 NO QW0045X5 NO
QW0045X6 NO QW0045X7 NO QW0045X8 NO
      13:30:38.62764700 922872 1 21 LOCK DETAIL 'BLANK'
      43.04726670 NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
-----
LOCK RES TYPE: DATA PAGE LOCK DBID: GLWBSQLP OBID: GLWSEMP RESOURCE ID: X'3000EF00'
```

```

IRLM FUNC CODE : LOCK (NAME)          RETURN TOKEN: X'7F69BF70'          REQUEST TOKEN      : X'00000000'
LOCK STATE      : SHARED              DB2 TOKEN      : X'00A4B00024C0A428'      IRLM RETURN CODE   :      4
LOCK ATTRIBUTES: NMODIFY NOFORCE      PROP TO XES    : NO              ASYN TO XES        : NO
LOCK DURATION   : MANUAL              REQUEST TYPE   :                  IRLM RETURN SUBCODE: B'0000100000000000'
PARENT TOKEN    : X'7F69C100'         GLOBAL/LOCAL: GLOBAL            OWNER              : 'BLANK'
CACHED STATE    : N/A                LOCK HASH VALUE : X'00C248EF'
QW0021CL: X'00'          QW0021U : X'00A900F420BCF4A8'  QW0021CT: X'7C278C00'  QW0021FL: B'00110000'
QW0021F3: B'00000000'    QW0021O : X'00A900F420BCF400'  QW0021IR: X'0000'     QW0021F2: B'00000000'
-----
13:30:38.62773847 922873 1 21 LOCK DETAIL 'BLANK'
43.04735564
NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
-----
LOCK RES TYPE: N/P                      NAME: N/P
IRLM FUNC CODE : UNLOCK (TOKEN)        RETURN TOKEN: X'00000000'        REQUEST TOKEN      : X'7F69BF70'
LOCK STATE      : SHARED              DB2 TOKEN      : X'00A4B00024C0A428'      IRLM RETURN CODE   :      0
LOCK ATTRIBUTES: NMODIFY NOFORCE      PROP TO XES    : NO              ASYN TO XES        : NO
LOCK DURATION   : MANUAL + 1          REQUEST TYPE   :                  IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN    : X'00000000'         GLOBAL/LOCAL: GLOBAL            OWNER              : 'BLANK'
CACHED STATE    : N/A                LOCK HASH VALUE : X'00C248EF'
QW0021CL: X'00'          QW0021U : X'00A900F420BCF4A8'  QW0021CT: X'7C278C00'  QW0021FL: B'00110000'
QW0021F3: B'00000000'    QW0021O : X'00A900F420BCF400'  QW0021IR: X'0000'     QW0021F2: B'00000000'
-----
LOCATION: DB9A                      OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)          PAGE: 1-248
GROUP: N/P                        RECORD TRACE - LONG                                     REQUESTED FROM: ALL 13:30:35.00
MEMBER: N/P                                                                TO: DATES          13:30:40.00
SUBSYSTEM: DB9A                                                           ACTUAL FROM: 08/07/09 13:30:35.09
DB2 VERSION: V9                                                           PAGE DATE: 08/07/09
PRIMAUTH CONNECT INSTANCE END_USER WS_NAME TRANSACT
ORIGAUTH CORRNAME CONNTYPE RECORD TIME DESTNO ACE IFC DESCRIPTION DATA
PLANNAME CORRNMBR TCB CPU TIME ID
-----
BART DB2CALL C49A0E034547 'BLANK' 'BLANK' 'BLANK'
BART BARTSQR1 DB2CALL 13:30:38.62788113 922874 1 58 END SQL <-- NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
DSNREXX 'BLANK' 43.04749483
-----
LOCATION NAME : DB9A
PKG COLLECTION ID : GLWBSQLP
PROGRAM NAME : EMPSEL
CONSISTENCY TOKEN : X'17BFB71C0BD53AB0'
STATEMENT NUMBER : 365 QUERY COMMAND ID : N/P
SQLCODE : 0 SQLSTATE: 00000
SQLERRP : DSN SQLTEXT : 00000
SQLWARN0: SQLWARN1: SQLWARN2: SQLWARN3:
SQLWARN4: SQLWARN5: SQLWARN6: SQLWARN7:
SQLWARN8: SQLWARN9: SQLWARNA:
SQLERRM:
-----
DATA TYPE INDX ROW PROC 365 ROW EXAM 392 STG1-QUAL 1 STG2-QUAL 1 ROW INSRT 0
ROW UPDTE 0 ROW DELET 0 PAGES 363 RI SCAN 0 RI DELET 0
LOB SCAN 0 LOB UPDTE 0
DATA TYPE SEQD ROW PROC 28254 ROW EXAM 28254 STG1-QUAL 0 STG2-QUAL 0 ROW INSRT 0
ROW UPDTE 0 ROW DELET 0 PAGES 1221 RI SCAN 0 RI DELET 0
LOB SCAN 0 LOB UPDTE 0
-----

```

As expected, job BARTSQR1 was resumed at 13:30:38.62763403. Note that this is a “normal” resume. This transaction was not selected as the deadlock victim, so it was able to continue as though nothing had happened, other than that it had to wait for the lock to become available.

BARTSQR1 tried to acquire an S lock for GLWBSQLP.GLWSEMP.X'3000EF00'. The lock request ends with an IRLM return code x'4' and subcode B'0000100000000000' or X'0800', which indicates that this lock was involved in a deadlock, but was not selected as a victim. Note that it took almost two seconds before the lock became available (and for DB2 to sort out the deadlock). This time will be reported as lock/latch wait time in the DB2 accounting record.

### The SQL statement that triggered the lock request

When you continue reviewing the RECTRACE report in Example 9-31 on page 292, you see that the LOCK request is followed by an UNLOCK request of the page (the unlock request is using the lock token that was returned on the LOCK request).

Next, an IFCID 58 record appears, which indicates that an SQL statement has ended. IFCID 58 has information about the SQL code of the statement, but also the program name (EMPSEL) and statement number (365). We can use the DB2 Admin Tool (Figure 9-3) to look up the statement text.

```

DB2 Admin ----- Extracted SQL ----- Columns 00001 00072
Command ==>                                         Scroll ==> CSR

***** ***** Top of Data *****
==MSG> -CAUTION- Use Make Data(MD) line command to
==MSG>         change NOTE lines to DATA lines
==MSG> -Warning- The UNDO command is not available until you change
==MSG>         your edit profile using the command RECOVERY ON.
=NOTE= -- SQL statements in  PACKAGE  : GLWBSQLP.EMPSEL.()
=NOTE= -- SQL in stmt: 242
000001 SELECT 0 INTO :H FROM SYSIBM.SYSDUMMY1 WITH UR
=NOTE= -- SQL in stmt: 273
000002 SELECT MAX (EMP_NO) INTO :H :H FROM GLWTEMP
=NOTE= -- SQL in stmt: 341
000003 SELECT MIN (EMP_NO) INTO :H :H FROM GLWTEMP WHERE EMP_NO >= :H :H
=NOTE= -- SQL in stmt: 365
000004 SELECT MIN (EMP_NO) INTO :H :H FROM GLWTEMP WHERE EMP_NO >= :H :H AND
000005 MANAGER = :H :H
=NOTE= -- SQL in stmt: 414
000006 SELECT MAX (EMP_NO) INTO :H :H FROM GLWTEMP WHERE EMP_NO <= :H :H
=NOTE= -- SQL in stmt: 438
000007 SELECT MAX (EMP_NO) INTO :H :H FROM GLWTEMP WHERE EMP_NO <= :H :H AND
000008 MANAGER = :H :H

```

Figure 9-3 Statement #365 of the EMPSEL package

We can also go back in the RECTRACE report and look for the start of SQL statement #365 (Example 9-32).

Example 9-32 Start of SELECT statement #365

13:30:36.65538864	921974	1	60	SELECT	--> 'BLANK'				
42.98594630				START	NETWORKID: USIBMSC	LUNAME: SCPDB9A	LWSEQ: 1		
					LOCATION NAME	: DB9A			
					PKG COLLECTION ID	: GLWBSQLP			
					PROGRAM NAME	: EMPSEL			
					CONSISTENCY TOKEN	: X'17BFB71COBD53AB0'			
					STATEMENT NUMBER	: 365			
					STATEMENT TYPE	: X'00'			
					QUERY COMMAND ID	: N/P			
					QUERY INSTANCE ID	: N/P			
					ISOLATION	: CS			
					REOPTIMIZATION	: NO			

The last part of the puzzle is to find the statement that acquired the X lock on PAGE GLWBSQLP.31.X'000243'.



First, we try to use the same technique we successfully used to find the other SQL statements and perform a search in the RECTRACE for a lock request on a page X'00024300' with a matching DBNAME and OBID. Although there are a number of lock requests for this page in the time frame covered by the RECTRACE report, they all happen to be S lock requests instead of X-lock requests (Example 9-33).

**Example 9-33** S-lock request for page x'243'

PRIMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME	TRANSACTION					
ORIGAUTH	CORRNAME	CONNTYPE	RECORD TIME	DESTNO ACE	IFC	DESCRIPTION	DATA			
PLANNAME	CORRNMBR		TCB CPU TIME	ID						
-----										
BART	DB2CALL	C49A0E034547	'BLANK'	'BLANK'			'BLANK'			
BART	BARTSQR1	DB2CALL	13:30:36.51560972	921576	1	21	LOCK DETAIL	NETWORKID:	USIBMSC LUNAME: SCPDB9A LUWSEQ: 1	
DSNREXX	'BLANK'		19.60176402							
-----										
LOCK RES TYPE: DATA PAGE LOCK				DBID: GLWBSQLP		OBID: 31		RESOURCE ID: X'00024300'		
IRLM FUNC CODE : LOCK (NAME)				RETURN TOKEN: X'FFFFFFF'		REQUEST TOKEN		: X'00000000'		
LOCK STATE : SHARED				DB2 TOKEN : X'00A4B00024C0A428'		IRLM RETURN CODE		: 4		
LOCK ATTRIBUTES: NMODIFY NOFORCE				PROP TO XES : NO		ASYN TO XES		: NO		
LOCK DURATION : MANUAL				REQUEST TYPE:		IRLM RETURN SUBCODE:		B'0001000000000000'		
PARENT TOKEN : X'7F69BEE0'				GLOBAL/LOCAL: GLOBAL		OWNER		: 'BLANK'		
CACHED STATE : N/A						LOCK HASH VALUE		: X'009CCA43'		
QW0021CL: X'00'		QW0021U : X'00A900F400000000'		QW0021CT: X'7C278C00'		QW0021FL: B'11110000'				
QW0021F3: B'00000001'		QW00210 : X'00A900F420BCF400'		QW0021IR: X'0000'		QW0021F2: B'00000000'				
-----										

To get a better idea of how much we need to expand the time frame to investigate when the X lock was acquired, we list the commit points that the job took. The X lock must have been requested after the previous commit, as a commit normally frees all X page locks.

To list the commit points of this job, we use the following RECTRACE command:

```
GLOBAL
  TIMEZONE(+4)
  INCLUDE (
    DB2ID(DB9A)
    IFCID(68,69,70,71,72,73,74,75,76,77,78,79,
      82,83,84,85,86,87,88,89)
    INSTANCE(C49A0E034547)
  )
RECTRACE
  TRACE
    LEVEL(LONG)
EXEC
```

An excerpt is shown in Example 9-34. We are looking for the unit of work that covers the time frame that includes the deadlock time (13:30:38.62062331).

**Example 9-34** Syncpoint records

PRIMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME	TRANSACT				
ORIGAUTH	CORRNAME	CONNTYPE	RECORD TIME	DESTNO ACE	DATA	DESCRIPTION			
PLANNAME	CORRNMBR		TCB CPU TIME	ID					
...									
BART	DB2CALL	C49A0E034547	13:30:34.76960472	919220	1	88	SYNC START -->	'BLANK'	
BART	BARTSQR1	DB2CALL	0.22525981					NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1	
DSNREXX	'BLANK'							PSWKEY X'80'	
								QW0088FR X'00795B20'	
			13:30:34.77155053	919247	1	89	SYNC END <--	'BLANK'	
			0.22537419					NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1	
								QW0089FR X'00795B20' QW0089RT 0	
								QW0089RS 0	
			13:30:41.20231217	933183	1	88	SYNC START -->	'BLANK'	
			0.22635466					NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1	
								PSWKEY X'80'	
...									

The X lock must have been acquired between 13:30:34.77 and 13:30:41.20. Actually, it must have been prior to the deadlock time, as we know that the lock was being held at that time, so we can reduce the time frame to 13:30:38.62. We can use the RECTRACE command with a different time frame, but we can also use a different technique.

The OMEGAMON PE locking trace has a number of options that allow you to only show lock requests for a specific object. This may not be as important for this case, as the trace is not extremely large, but it can be extremely valuable when you have to handle large trace data sets or when the time frame in which the lock was acquired is very large. The following OMEGAMON PE command can be used to limit the report to only list the datapage lock requests related to the GLWBSQLP database and pageset (OBID) 31:

```
DB2PM
GLOBAL
    TIMEZONE (+4)
    FROM(,13:30:34.77)
    TO(,13:30:41.20)
    INCLUDE (
        DB2ID(DB9A)
    )
LOCKING
    TRACE
    INCLUDE (
        INSTANCE(C49A0E034547)
        DATABASE(GLWBSQLP)
        PAGESET(31)
        RESOURCETYPE(DATAPAGE)
    )
    LEVEL(DETAIL)
EXEC
```

An excerpt of the report can be found in Example 9-35. It shows that the X lock was acquired at 13:30:34.79896342, just prior to our first interval.

Example 9-35 LOCK trace for GLWBSQLP.31.x'000243'

PRIMAUTH	CORRNAME	CONNTYPE	EVENT	TIMESTAMP	---	LOCK	RESOURCE	---	EVENT	SPECIFIC	DATA
ORIGAUTH	CORRNMBR	INSTANCE	RELATED	TIMESTAMP	EVENT	TYPE	NAME				
PLANNAME	CONNECT										
BART	BARTSQR1	DB2CALL	13:30:34.79896342		LOCK	DATAPAGE	DB =GLWBSQLP		DURATION=COMMIT	STATE=X	
BART	'BLANK'	C49A0E034547			REQUEST		OB =31		RSN CODE= 0	RTNCD= 0	
DSNREXX	DB2CALL						PAGE=X'000243'		HASH =X'009CCA43'		
			13:30:34.80626317		LOCK	DATAPAGE	DB =GLWBSQLP		DURATION=COMMIT	STATE=S	
					REQUEST		OB =31		RSN CODE=X'10'	RTNCD= 4	
							PAGE=X'000243'		HASH =X'009CCA43'		
			13:30:34.86173184		LOCK	DATAPAGE	DB =GLWBSQLP		DURATION=MANUAL	STATE=S	
					REQUEST		OB =31		RSN CODE=X'10'	RTNCD= 4	
							PAGE=X'000243'		HASH =X'009CCA43'		
			13:30:34.98859446		LOCK	DATAPAGE	DB =GLWBSQLP		DURATION=MANUAL	STATE=S	
					REQUEST		OB =31		RSN CODE=X'10'	RTNCD= 4	
							PAGE=X'000243'		HASH =X'009CCA43'		
....											

So now we can better target the time frame for our RECTRACE request to identify the SQL statement that acquired the X lock as follows:

```
DB2PM
GLOBAL
  TIMEZONE (+4)
  FROM(,13:30:34.77)
  TO(,13:30:34.80)
  INCLUDE (
    DB2ID(DB9A)
    INSTANCE(C49A0E034547)
  )
RECTRACE
TRACE
  LEVEL(LONG)
EXEC
```

An excerpt of the RECTRACE report is shown in Example 9-36.

**Example 9-36** RECTRACE of SQL statement acquiring the X-lock

PRIMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME	TRANSACT									
ORIGAUTH	CORRNAME	CONNTYPE	RECORD TIME	DESTNO ACE	IFC	DESCRIPTION	DATA							
PLANNAME	CORRNMBR		TCB CPU TIME	ID										
-----														
BART	DB2CALL	C49A0E034547	'BLANK'	'BLANK'										
BART	BARTSQR1	DB2CALL	13:30:34.79886947	919362	1	61	INSERT	---	NETWORKID: USIBMSC	LUNAME: SCPDB9A LUWSEQ: 1				
DSNREXX	'BLANK'		19.58008561				START		LOCATION NAME : DB9A					
									PKG COLLECTION ID : GLWBSQLP					
									PROGRAM NAME : PRJADD					
									CONSISTENCY TOKEN : X'17BFB6FD1A8A75BC'					
									STATEMENT NUMBER : 1024					
									STATEMENT TYPE : INSERT TYPE					
									CURSOR NAME : N/P					
									QUERY COMMAND ID : N/P					
									QUERY INSTANCE ID : N/P					
									ISOLATION : CS					
									REOPTIMIZATION : NO					
									'BLANK'					
									NETWORKID: USIBMSC	LUNAME: SCPDB9A LUWSEQ: 1				
-----														
LOCK RES TYPE: PAGESET LOCK DBID: GLWBSQLP OBID: GLWSPRJ														
IRLM FUNC CODE : LOCK (NAME)				RETURN TOKEN: X'7F69BEE0'				REQUEST TOKEN : X'00000000'						
LOCK STATE : INTENT EXCLUSIVE				DB2 TOKEN : X'00A4B00024C0A428'				IRLM RETURN CODE : 4						
LOCK ATTRIBUTES: MODIFY NOFORCE				PROP TO XES : NO				ASYN TO XES : NO						
LOCK DURATION : COMMIT				REQUEST TYPE:				IRLM RETURN SUBCODE: B'0010000000000000'						
PARENT TOKEN : X'00000000'				GLOBAL/LOCAL: GLOBAL				OWNER : 'BLANK'						
CACHED STATE : N/A								LOCK HASH VALUE : X'00021EF2'						
QW0021CL: X'00'				QW0021U : X'00A900F400000000'				QW0021CT: X'7C278C00'						
QW0021F3: B'00000000'				QW0021O : X'00A900F420BCF400'				QW0021IR: X'0000'						
								QW0021FL: B'00110010'						
								QW0021F2: B'00000000'						
-----														
									13:30:34.79889419	919364	1	21	LOCK DETAIL	'BLANK'
									19.58010444				NETWORKID: USIBMSC	LUNAME: SCPDB9A LUWSEQ: 1
-----														
LOCK RES TYPE: TABLE LOCK DBID: GLWBSQLP OBID: 31														
IRLM FUNC CODE : LOCK (NAME)				RETURN TOKEN: X'7F69BFD0'				REQUEST TOKEN : X'00000000'						
LOCK STATE : INTENT EXCLUSIVE				DB2 TOKEN : X'00A4B00024C0A428'				IRLM RETURN CODE : 4						
LOCK ATTRIBUTES: MODIFY NOFORCE				PROP TO XES : NO				ASYN TO XES : NO						
LOCK DURATION : COMMIT				REQUEST TYPE:				IRLM RETURN SUBCODE: B'0010000000000000'						
PARENT TOKEN : X'00000000'				GLOBAL/LOCAL: GLOBAL				OWNER : 'BLANK'						
CACHED STATE : N/A								LOCK HASH VALUE : X'00021FF2'						
QW0021CL: X'00'				QW0021U : X'00A900F400000000'				QW0021CT: X'7C278C00'						
QW0021F3: B'00000000'				QW0021O : X'00A900F420BCF400'				QW0021IR: X'0000'						
								QW0021FL: B'00110010'						
								QW0021F2: B'00000000'						
-----														
									13:30:34.79889419	919365	1	211	CLAIM DATA	'BLANK'
									19.58013255				NETWORKID: USIBMSC	LUNAME: SCPDB9A LUWSEQ: 1
-----														
DBID: GLWBSQLP PSID: GLWSPRJ PARTITION NO.: 0 CLAIM REQUEST TYPE: ACQUIRE CLAIM CLASS: CS READ														
CLAIM DURATION: HELD UNTIL COMMIT CLAIM RESULT: SUCCESSFUL														
-----														
BART	DB2CALL	C49A0E034547	'BLANK'	'BLANK'										
BART	BARTSQR1	DB2CALL	13:30:34.79892975	919366	1	211	CLAIM DATA		NETWORKID: USIBMSC	LUNAME: SCPDB9A LUWSEQ: 1				
DSNREXX	'BLANK'		19.58013528											
-----														
DBID: GLWBSQLP PSID: GLWSPRJ PARTITION NO.: 0 CLAIM REQUEST TYPE: ACQUIRE CLAIM CLASS: WRITE														

```

CLAIM DURATION: HELD UNTIL COMMIT CLAIM RESULT: SUCCESSFUL
-----
13:30:34.79894070 919367 1 211 CLAIM DATA 'BLANK'
19.58014395 NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
-----
DBID: GLWBSQLP PSID: GLWXPJ1 PARTITION NO.: 0 CLAIM REQUEST TYPE: ACQUIRE CLAIM CLASS: CS READ
CLAIM DURATION: HELD UNTIL COMMIT CLAIM RESULT: SUCCESSFUL
-----
13:30:34.79894537 919368 1 211 CLAIM DATA 'BLANK'
19.58014632 NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
-----
DBID: GLWBSQLP PSID: GLWXPJ1 PARTITION NO.: 0 CLAIM REQUEST TYPE: ACQUIRE CLAIM CLASS: WRITE
CLAIM DURATION: HELD UNTIL COMMIT CLAIM RESULT: SUCCESSFUL
-----
13:30:34.79896342 919369 1 21 LOCK DETAIL 'BLANK'
19.58016214 NETWORKID: USIBMSC LUNAME: SCPDB9A LUWSEQ: 1
-----
LOCK RES TYPE: DATA PAGE LOCK DBID: GLWBSQLP OBID: 31 RESOURCE ID: X'00024300'
IRLM FUNC CODE : LOCK (NAME) RETURN TOKEN: X'7F6ABCA0' REQUEST TOKEN : X'00000000'
LOCK STATE : EXCLUSIVE DB2 TOKEN : X'00A4B00024C0A428' IRLM RETURN CODE : 0
LOCK ATTRIBUTES: MODIFY NOFORCE PROP TO XES : NO ASYN TO XES : NO
LOCK DURATION : COMMIT REQUEST TYPE: IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN : X'7F69BEE0' GLOBAL/LOCAL: GLOBAL OWNER : 'BLANK'
CACHED STATE : N/A LOCK HASH VALUE : X'009CCA43'
QW0021CL: X'00' QW0021U : X'00A900F400000000' QW0021CT: X'7C278C00' QW0021FL: B'10110010'
QW0021F3: B'00000000' QW0021O : X'00A900F420BCF400' QW0021IR: X'0000' QW0021F2: B'00000000'
-----

```

We look for the time stamp (for when the X lock was acquired) that we obtained from the locking trace report shown in Example 9-35 on page 296. Then we search backwards to verify which SQL statement issued the X lock request.

The statement shows an INSERT that is part of the PRJADD package. The statement number is 1024. We can look up the statement text using the DB2 Admin Tool (Example 9-37).

#### Example 9-37 INSERT statement #1024

```

DB2 Admin ----- Extracted SQL ----- Columns 00001 00072
Command ==> Scroll ==> CSR

```

```

***** ***** Top of Data *****
=NOTE= -- SQL statements in PACKAGE : GLWBSQLP.PRJADD.()
=NOTE= -- SQL in stmt: 471
000001 SELECT 0 INTO :H FROM SYSIBM.SYSDUMMY1 WITH UR
=NOTE= -- SQL in stmt: 533
000002 DECLARE C1 CURSOR FOR SELECT ACT_NO, ACTKWD, ACTDESC FROM GLWTACT ORDER
000003 BY ACT_NO
=NOTE= -- SQL in stmt: 579
000004 CALL SPLANO (:H:H, :H :H)
=NOTE= -- SQL in stmt: 686
000005 CALL DPTSEL (:H :H, :H :H, :H :H, :H :H, :H :H, :H :H)
=NOTE= -- SQL in stmt: 782
000006 CALL PRJANO (:H :H, :H :H)
=NOTE= -- SQL in stmt: 933
000007 SELECT MIN (EMP_NO) INTO :H :H FROM GLWTEMP WHERE BIRTHDATE = (SELECT
000008 MIN (BIRTHDATE) FROM GLWTEMP WHERE WORKDEPT = :H :H)
=NOTE= -- SQL in stmt: 1024
000009 INSERT INTO GLWTPRJ (PROJ_NO, PROJNAME, DEPT_NO, RESP_EMP_NO, PRSTDATE,
000010 PRENDATE, PROJ_DESC, CREATED_BY, UPDATED_BY) VALUES (:H :H, :H :H, :H
000011 :H, :H :H, :H :H, :H :H, :H :H, USER, USER)

```

## 9.2.5 The complete deadlock picture

Now that we have viewed all the locks and statements involved, we have the complete picture of the deadlock, as shown in Figure 9-4.

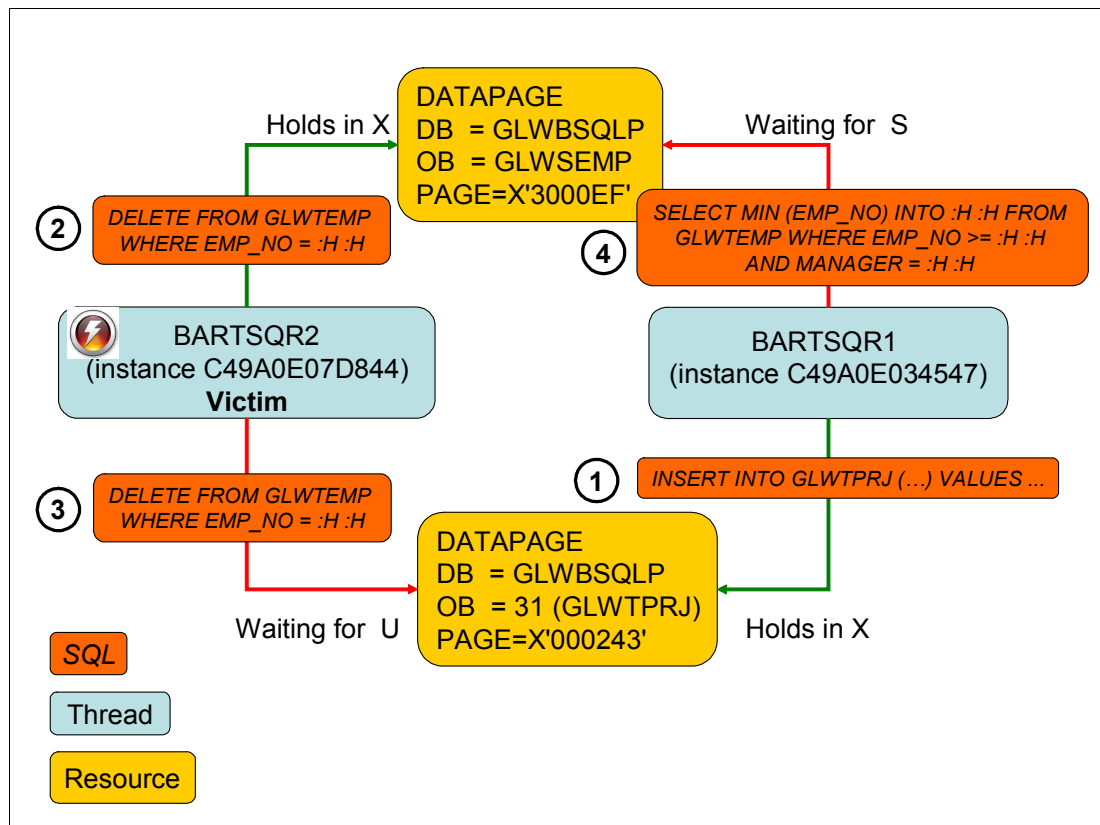


Figure 9-4 Complete deadlock picture

It shows all the transactions involved, the resources involved, as well as the SQL statements. The sequence of events shows that both transactions access both objects in a different sequence. BARTSQR2 accesses GLWTEMP first, then GLWTPRJ. BARTSQR1 accesses GLWTPRJ first, then GLWTEMP. Accessing objects in a different sequence often results in deadlocks; refer to 5.19, “How to prevent locking problems” on page 150 for more information on this topic.

Building the full picture of a locking conflict can be time consuming and is not always necessary. As you become more experienced with diagnosing locking conflicts, you will be able to easily discover where the potential problem areas are, and you can immediately zoom in without constructing the full picture when you spot a problem.

In “The SQL statement that triggered the deadlock” on page 289, we noticed that `DELETE FROM GLWTEMP` not only results in locks on GLWTEMP, which is clearly expected, but also acquires U locks on the GLWTPRJ table. This seems unexpected since this is a different table.

If you encounter such a case, you should investigate it immediately, as this is most likely going to be the cause of the problem. But as we wanted to illustrate the process, we constructed the full deadlock picture first before zooming in on the problem.

Let us now look at the definition of the GLWTEMP table using the DB2 Admin Tool.  
 Example 9-38 shows the GLWTEMP table. By using the RI command, you can list the referential relationships.

Example 9-38 GLWTEMP table

```
DB2 Admin ----- DB9A Tables, Views, and Aliases ---- Row 1 to 1 of 1
Command ==> Scroll ==> CSR

Commands: GRANT MIG
Line commands:
C - Columns A - Auth L - List X - Indexes S - Table space D - Database
V - Views T - Tables P - Plans Y - Synonyms SEL - Select prototyping
? - Show all line commands

Sel      Name                Schema      T DB Name    TS Name      Cols      Rows Checks
-----
RI      GLWTEMP                GLWBSQLP T GLWBSQLP  GLWSEMP      19      30000      0
```

Example 9-39 shows all the referential relationships that GLWTEMP is involved in, directly or indirectly. GLWTEMP is a parent for GLWTPRJ (child relationship).

Example 9-39 RI relationships of GLWTEMP

```

DB2 Admin ----- DB9A Referential Integrity Constraints Row 1 to 14 of 14
Command ==> Scroll ==> CSR

Line commands:
T - Tables      FC - Parent Key Columns      TC - Child Key Columns

  Rel Refers      Refers      Relation D Col      Origin      Origin
Sel Typ Owner      Name      Name      R Count      Owner      Table
  *  *  *          *          *          *          *  *          *
----->----->----->----->----->----->----->----->----->
  CHR GLWBSQLP GLWTDPT      GLWFDPT1 N      1 GLWBSQLP GLWTEMP
  CHR GLWBSQLP GLWTEPA      GLWFEP1 C      1 GLWBSQLP GLWTEMP
tc  CHR GLWBSQLP GLWTPRJ      GLWFPRJ2 N      1 GLWBSQLP GLWTEMP
  PAR GLWBSQLP GLWTDPT      GLWFEMP1 N      1 GLWBSQLP GLWTEMP
  CHR GLWBSQLP GLWTPRJ      GLWFPRJ1 C      1 GLWBSQLP GLWTDPT
  CHR GLWBSQLP GLWTEMP      GLWFEMP1 N      1 GLWBSQLP GLWTDPT
  PAR GLWBSQLP GLWTEMP      GLWFDPT1 N      1 GLWBSQLP GLWTDPT
  PAR GLWBSQLP GLWTEMP      GLWFEP1 C      1 GLWBSQLP GLWTEPA
  PAR GLWBSQLP GLWTPJA      GLWFEP2 C      2 GLWBSQLP GLWTEPA
  CHR GLWBSQLP GLWTPJA      GLWFPJA1 C      1 GLWBSQLP GLWTPRJ
  PAR GLWBSQLP GLWTDPT      GLWFPRJ1 C      1 GLWBSQLP GLWTPRJ
  PAR GLWBSQLP GLWTEMP      GLWFPRJ2 N      1 GLWBSQLP GLWTPRJ
  CHR GLWBSQLP GLWTEPA      GLWFEP2 C      2 GLWBSQLP GLWTPJA
  PAR GLWBSQLP GLWTPRJ      GLWFPJA1 C      1 GLWBSQLP GLWTPJA

```

By using the TC command, you can list the child key column(s), as shown in Example 9-40.

*Example 9-40 FK column in GLWTPRJ to GLWTEMP*

---

```
DB2 Admin ----- DB9A Columns ----- Row 1 to 1 of 1
Command ==>                                     Scroll ==> CSR
```

Line commands:

```
T - Tables  ST - Specific table  A - Auth  GR - Grant  SEQ -Identity column
H - Homonyms  I - Interpret  UR - Update runstats  LAB - Label  COM - Comment
DI - Distribution stats  PST - Partition stats  E - Source data type
X - Indexes  SX - Specific indexes  RH - Runstats history  CON - Constraints
```

---

Sel	Schema	Name	Column Name	Col No	Col Type	Length	N	D	F
*	*	*	*	*	*	*	*	*	*
----	-----	-----	-----	-----	-----	-----	-----	-----	-----
	GLWSQLP	GLWTPRJ	RESP_EMP_NO	4	INTEGER		4	Y	Y N

---

Looking at the DDL that was used to create our tables (Example 9-41) confirms that there is a foreign key (GLWFPRJ2) under the RESP\_EMP\_NO column of the GLWTPRJ that refers to the EMP\_NO column in GLWTEMP.

*Example 9-41 GLWFPRJ2 foreign key relationship*

---

```
ALTER TABLE GLWSAMP.GLWTPRJ FOREIGN KEY GLWFPRJ2
      (RESP_EMP_NO)
REFERENCES GLWSAMP.GLWTEMP
      (EMP_NO)
ON DELETE SET NULL ;
```

---

When deleting rows from the GLWTEMP table, DB2 has to check the GLWTPRJ table for matching employee numbers, and set the value of the RESP\_EMP\_NO to nulls if we find a matching row. This explains why a DELETE of an employee from the GLWTEMP table leads to accessing the GLWTPRJ table.

In a locking trace for the DELETE statement (Example 9-42 on page 302), you can see that we first acquire an X lock on a page in the GLWSEMP table space, which is expected because we are deleting a row from the GLWTEMP table, and then there are many lock requests (we only show a few here) for OBID 31, the GLWTPRJ table. So we scan the GLWTPRJ table and check the RESP\_EMP\_NO column to see whether it matches the EMP\_NO that we are deleting from the GLWTEMP table, and if it does, set it to null.

As there are other processes accessing GLWPRJ and another deleter from GLWTEMP, scanning the GLWPRJ table impacts performance and concurrency.

**Example 9-42 Lock requests issued by the DELETE FROM GLWTEMP statement**

GROUP: N/P			LOCKING TRACE - DETAIL				REQUESTED FROM: ALL		13:30:36.71		
MEMBER: N/P							TO: DATES		13:30:41.20		
SUBSYSTEM: DB9A							ACTUAL FROM: 08/07/09		13:30:36.71		
DB2 VERSION: V9			SCOPE: MEMBER				PAGE DATE: 08/07/09				
PRIMAUTH	CORRNAME	CONNTYPE	EVENT TIMESTAMP		---		L O C K		R E S O U R C E ---		
ORIGAUTH	CORRNMBR	INSTANCE	RELATED	TIMESTAMP	EVENT	TYPE	NAME		EVENT SPECIFIC DATA		
PLANNAME	CONNECT										
-----			-----								
BART	BARTSQR2	DB2CALL	13:30:36.71216898	LOCK	DATAPAGE	DB	=GLWBSQLP		DURATION=COMMIT STATE=X		
BART	'BLANK'	C49A0E07D844		REQUEST		OB	=GLWSEMP		RSN CODE= 0 RTNCD= 0		
DSNREXX	DB2CALL					PAGE	=X'3000EF'		HASH =X'00C248EF'		
			13:30:36.71238775	LOCK	DATAPAGE	DB	=GLWBSQLP		DURATION=MANUAL STATE=U		
				REQUEST		OB	=31		RSN CODE= 0 RTNCD= 0		
						PAGE	=X'000002'		HASH =X'000C8802'		
			13:30:36.71239883	LOCK	DATAPAGE	DB	=GLWBSQLP		DURATION=MANUAL STATE=U		
				REQUEST		OB	=31		RSN CODE= 0 RTNCD= 0		
						PAGE	=X'0001A1'		HASH =X'006449A1'		
.....											
BART	BARTSQR2	DB2CALL	13:30:36.71856989	LOCK	DATAPAGE	DB	=GLWBSQLP		DURATION=MANUAL STATE=U		
BART	'BLANK'	C49A0E07D844		REQUEST		OB	=31		RSN CODE= 0 RTNCD= 0		
DSNREXX	DB2CALL					PAGE	=X'000242'		HASH =X'009C8A42'		
			13:30:36.71860151	LOCK	DATAPAGE	DB	=GLWBSQLP		DURATION=MANUAL STATE=U		
				SUSPEND		OB	=31		ORIG.RSN=LATCH CONT GENER		
						PAGE	=X'000243'		HASH =X'009CCA43'		
			13:30:38.61881981	LOCK	DATAPAGE	DB	=GLWBSQLP		SUSP.TIME =1.900218 LOCAL CONTENTION=Y*		
			13:30:36.71860151	RESUME		OB	=31		DURATION =MANUAL LATCH CONTENTION=N		
						PAGE	=X'000243'		STATE =U IRLM QUEUED REQ =N		
									RESUME RSN=DEADLOCK		
									HASH =X'009CCA43'		

The OMEGAMON PE input statement for this report is:

```

DB2PM
TIMEZONE (+4)
    FROM(,13:30:36.71)
    TO(,13:30:41.20)
    INCLUDE (
        DB2ID(DB9A)
        INSTANCE(C49A0E07D844)
    )
LOCKING
TRACE
    INCLUDE (
        INSTANCE(C49A0E07D844)
        DATABASE(GLWBSQLP)
        PAGESET(31, GLWSEMP)
        RESOURCETYPE(DATAPAGE)
    )
    LEVEL(DETAIL)
EXEC

```

Creating an index under the RESP\_EMP\_NO foreign key column will boost performance and reduce the chances of a deadlock.

In a second test, after creating this index (and some other indexes), there were no deadlocks, and the lock/latch time of the jobs was drastically reduced.





# Part 4

## Data sharing

In this part, we focus on considerations related to serialization in a data sharing environment. Most of the considerations in the other parts of this publication still apply and should be considered as prerequisites for the considerations in this part.

This part contains the following chapters:

- ▶ Chapter 10, “Global locking” on page 305
- ▶ Chapter 11, “Monitoring data sharing locking activity” on page 365
- ▶ Chapter 12, “Database and application design in data sharing” on page 417





## Global locking

DB2 data sharing uses the global locking mechanism to serialize shared resources between different DB2 members of a data sharing group. This chapter describes how global locking works in a DB2 data sharing environment.

The following topics are covered:

- ▶ Global locking overview
- ▶ DB2 data sharing system locking components
- ▶ L-locks versus P-locks
- ▶ How inter-DB2 data coherency or read-write interest works
- ▶ Data sharing locking optimizations
- ▶ Page P-locks
- ▶ Pseudo-close and the GBP-dependency
- ▶ Timeout and deadlock in a data sharing environment
- ▶ Pseudo-close and the GBP-dependency
- ▶ Global contention
- ▶ Lock avoidance in data sharing

## 10.1 Global locking overview

DB2 data sharing improves the availability of DB2 data, extends the processing capacity of your system, provides more flexible ways to configure your environment, and allows for increased transaction rates. It provides the highest level of scalability, performance, and continuous availability to enterprise applications that use DB2 data concurrently.

In DB2 data sharing, data can be shared by DB2 subsystems that are members of the data sharing group. Sharing of data by more than one DB2 subsystem raises issues of maintaining data integrity while having inter-system concurrency. For data integrity, DB2 uses a *global locking* mechanism for concurrency control and a buffer coherency control mechanism to provide data consistency between different members in a data sharing environment.

The following are factors that must be addressed and resolved to preserve data integrity in a multi-system data sharing environment:

- ▶ Concurrency control (global locking)
- ▶ Inter-system cache coherency

The concurrency control in DB2 data sharing exists both within a specific member and among all members of a data sharing group. DB2 continues to use IRLM for all lock requests and the locks are (almost) all *global locks*, that is, the scope of the serialization control is across all DB2 data sharing members, as opposed to a non-data sharing configuration, where all locks managed by IRLM have a *single-system scope*.

Cache coherency control, or managing changed data, addresses the situation in which one DB2 member changes data rows that already reside in the buffers of other members. In this publication, we focus on concurrency control and the physical locking mechanism in DB2 data sharing. For more information about cache or buffer coherency control, refer to *DB2 Version 9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845.

A *global lock* (also called a data sharing lock) is one that provides serialization across the data sharing group, that is, one that a DB2 data sharing member has to make known to other members (subsystems) in the data sharing group.

A *local lock* provides serialization within a single DB2 only (the same kind you get in a non-data sharing DB2 subsystem), and you still get a few in a data sharing environment.

Figure 10-1 introduces the DB2 data sharing environment that will be used in throughout this chapter. The DB2 data sharing group DB9CG has two DB2 members: D9C1 and D9C2. The IRLM names for the DB2 members is I9C1 and I9C2, respectively. Actually, the DB9CG is a three member data sharing group, but the third member was not used in our test environment and was permanently quiesced.

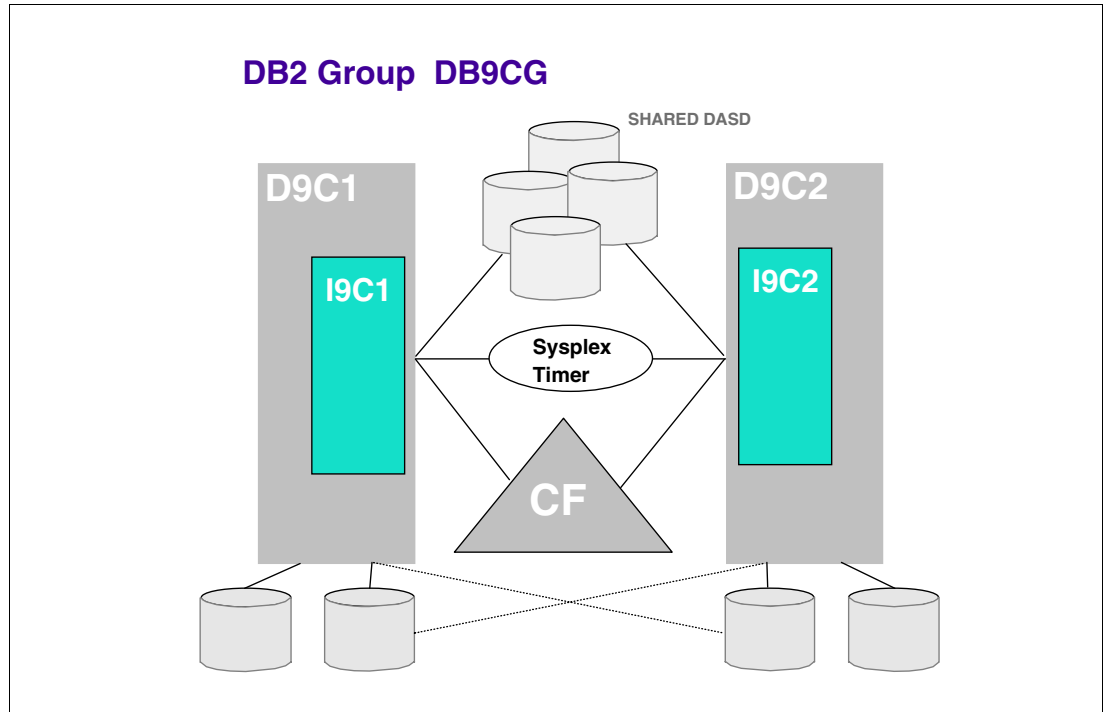


Figure 10-1 DB9CG DB2 data sharing group

DB2 data sharing is based on the premise that all DB2 data is shareable. Thus, any lock that is taken by a transaction on a database resource (table space, table, page, row, and so on) is a global lock because the database resource on which that lock is held has the potential of being accessed from multiple DB2 members.

When a global lock is requested (for example, on a table space page), DB2 interfaces with its associated IRLM, as it would normally do in a single-system environment. IRLM then checks its local structures to determine whether or not the lock is locally grantable, that is, whether or not the lock request can be granted based on the local IRLM's knowledge of the locking information. If the lock is locally grantable, then IRLM again checks its local structures to determine if the lock needs to be propagated to the coupling facility lock structure for inter-DB2 lock compatibility checking.

Figure 10-2 presents an general overview of the DB2 data sharing global locking flow. The example shown in Figure 10-2 explains global locking at the conceptual level. It is only intended to demonstrate how communication is performed at a high level, and therefore it contains a number of inaccuracies, but they are not relevant for the high level picture.

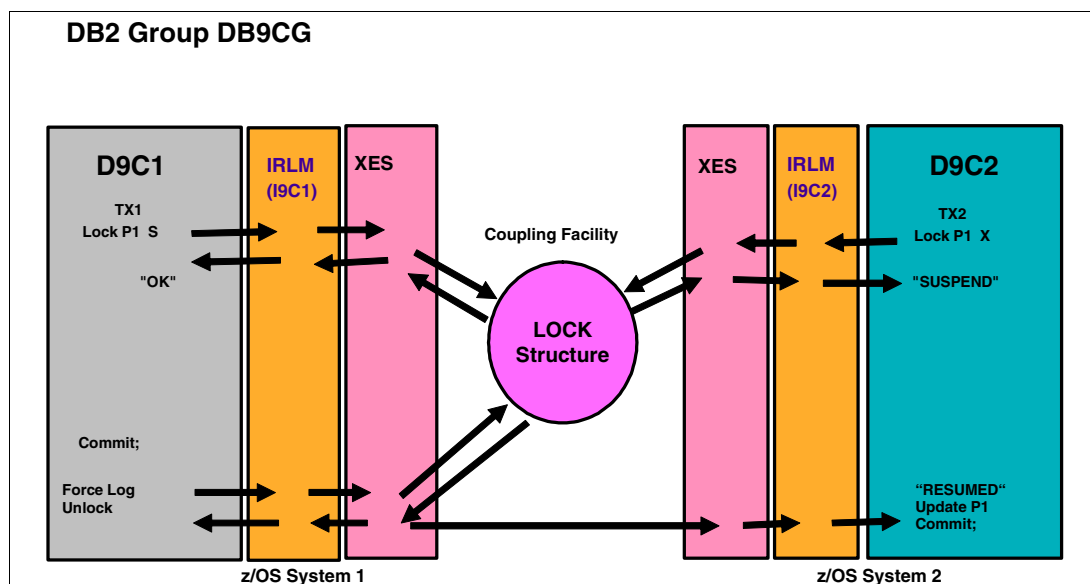


Figure 10-2 Data sharing global locking flow

The transaction Tx1 in DB2 member D9C1 reads page P1 and requests an S Lock on that page. In this case, transaction Tx1 is granted a S Lock on page P1, as no interest from another member was found. To verify this situation, IRLM I9C1R1 uses the he system lock manager (SLM), a component of the z/OS cross-system extended services (XES) to access the lock structure in the coupling facility to check and see if the other IRLM I9C2 has a global lock or interest on page P1. As this is not the case, the S lock will be granted, and IRLM I9C1 writes this lock to the coupling facility lock structure through XES.

Transaction Tx2 starts on member D9C2 and wants to update page P1 and requests an X lock. IRLM I9C2 looks into the coupling facility lock structure and finds that the requested page already has an S lock, and therefore an X lock cannot be granted, as they are incompatible.

Then transaction Tx2 will be suspended until transaction Tx1 commits and release the S lock on page P1.

Transaction Tx2 is resumed and can now proceed to update page P1 and acquire the X lock.

In a data sharing environment, locks can no longer be granted by the local IRLM. The different IRLMs have to communicate with each other by way of XES (SLM) and the coupling facility before a lock can be granted to make sure none of the other members already hold the lock in an incompatible state.

## 10.2 DB2 data sharing system locking components

Internal Resource Lock Manager (IRLM) assumes responsibility for global locking in a data sharing environment. Each IRLM in a data sharing group continues to manage the locks locally within the DB2 member. In addition, the IRLMs process global lock requests using the cross-system extended services (XES) component of z/OS to communicate with the coupling facility (CF) and to manage the lock structure and perform global locking. With the current technology, this interaction between IRLM and the coupling facility using XES to communicate happens in a matter of microseconds.

Figure 10-3 presents the z/OS and DB2 system main components used by global locking in a DB2 data sharing environment.

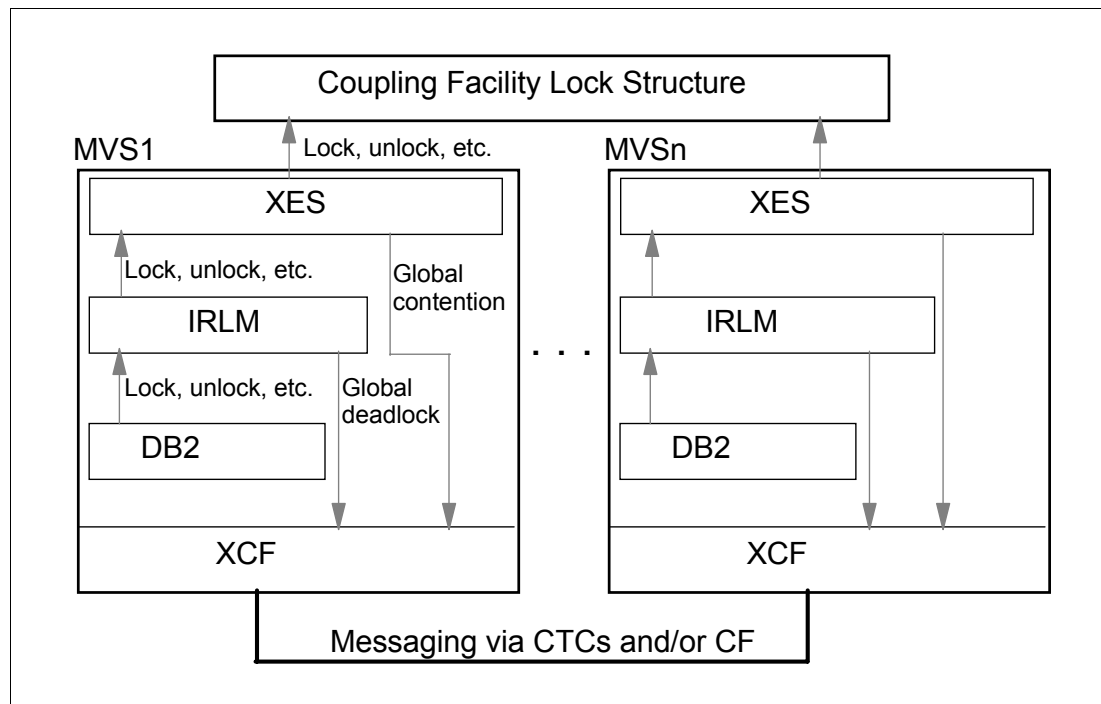


Figure 10-3 DB2 data sharing global locking system components

In global locking environment, it is important to understand that when IRLM receives a global lock request from DB2, IRLM determines whether or not the lock needs to be propagated (sent to XES). IRLM uses a number of important optimizations to suppress the propagation of the global lock to XES when it is not needed. Refer to 10.6, “Data sharing locking optimizations” on page 322 for details about global lock propagation.

If IRLM determines that lock propagation is needed, then IRLM in turn calls the MVS XES component. The system lock manager (SLM), a component of the z/OS cross-system extended services (XES), presents global lock information to the lock structure on behalf of each member's IRLM. For simplicity's sake, we use XES to refer to the SLM lock manager in this book.

XES uses its own software control block structures and the coupling facility lock structure to quickly determine if inter-DB2 lock contention may exist. If no inter-DB2 contention is detected, then the lock is granted synchronously to the executor of the requesting transaction. If XES does detect possible inter-DB2 lock contention, then the transaction is suspended, and XES uses MVS Cross-system Coupling Facility (XCF) signalling to exchange information between the IRLMs which, in most cases, resolves the contention. Refer to 10.10, “Global contention” on page 353 for more information about how data sharing handles the inter-DB2 contention process. IRLM also uses XCF services for things such as global deadlock detection.

Generally speaking, you can think of IRLM as the “local lock manager” (LLM) and XES in conjunction with the CF lock structure as the “global lock manager” (GLM). This is an over-simplification, because IRLM also incorporates many GLM functions (such as global deadlock detection), but the LLM and GLM concepts can help clarify how things work. Refer to 10.10.1, “Global lock manager” on page 353 for more information about GLM.

IRLM is a very robust lock manager that has matured over the course of decades. IRLM supports 11 lock compatibility states (eight of which are used by DB2; two of these, “US” and “NSU”, are used only for data sharing P-locks). In addition, IRLM supports many lock attributes, such as “duration” and understands “parent/child” locks.

In contrast, XES and the coupling facility lock structure do *not* support many different lock compatibility states or lock attributes. XES/CF supports only two lock compatibility states: share (S) and exclusive (X). The main purpose of the lock management functions of XES and the coupling facility is to provide a fast switch to quickly detect possible inter-DB2 lock contention.

Table 10-1 shows the main distinctions between IRLM and XES locking components.

Table 10-1 IRLM and XES Locking differences

IRLM	XES
A lock maps a “resource” to a “transaction”.	A lock maps a “resource” to a DB2 member.
Understands 11 lock compatibility states.	Understands two lock compatibility states.
Many lock attributes supported.	No attributes supported.
Timeout/deadlock supported.	No timeout/deadlock support.
Understands the difference between P-locks and L-locks. <sup>a</sup>	Cannot distinguish between P-locks and L-locks.

a. See 10.4, “L-locks versus P-locks” on page 314 for more information about P-locks and L-locks.

Table 10-2 shows the mapping between the IRLM parent L-lock modes and the XES lock modes. (Note that the mapping changed to this configuration with the DB2 V8 NFM locking protocol 2.)

Table 10-2 IRLM parent L-lock to XES lock mode mapping

IRLM parent L-lock mode	XES lock mode
IS, IX	S
S, U, SIX, and X	X



Table 10-3 shows the mapping between the IRLM page set P-lock modes and their XES lock modes.

*Table 10-3 IRLM page set P-lock to XES lock mode mapping*

IRLM page set P-lock mode	XES lock mode
IS, S	S
IX, SIX, X, and NSU <sup>a</sup>	X

a. NSU stands for “non-shared update”. It acts like an X lock, but is only used during P-lock negotiation from an X to an SIX.

Child (page and row) L-lock and page P-lock modes map to XES lock modes as shown in Table 10-4.

*Table 10-4 IRLM child L-lock and page P-lock to XES lock mode mapping*

IRLM child L-lock or page P-lock mode	XES lock mode
S	S
U and X	X

This publication assumes DB2 9 for z/OS and locking protocol 2. Locking protocol 2 was introduced in DB2 V8 NFM, after a group wide restart of the data sharing group. Locking protocol 2 avoids the cost of global contention processing for table space level locks. It also improves availability due to a reduction in the number retained locks that will have to be acquired after a DB2 subsystem or z/OS system failure. For more details, see Chapter 8, “DB2 UDB for z/OS Version 8 Performance Topics, SG24-6465.

## 10.3 Introducing the DB2 lock structure

A lock structure is one of the three types of Coupling Facility (CF) structures (the other two are “cache” and “list” structures). A structure is a logical construct used by MVS to map and manage storage in the coupling facility. A coupling facility can hold one or more structures of any type; however, each structure must reside within a single coupling facility. In addition to the lock structure, which is used for global locking, DB2 data sharing uses a list structure for the Shared Communications Area (SCA) and cache structures for the group buffer pools (GBps).

An overview of the CF structures that are used by DB2 is shown in Figure 10-4.

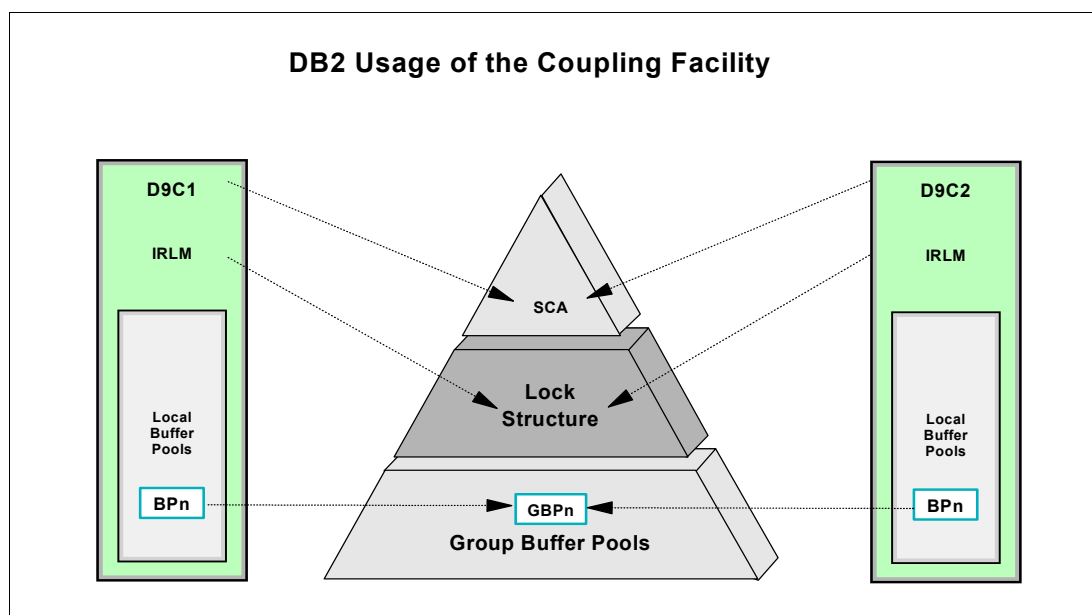


Figure 10-4 Overview of CF structures used by DB2

All lock information that members in the group have to share is kept in the lock structure in the coupling facility. The name of the lock structure for DB2 is always as follows:

*DB2-groupname\_LOCK1*

The *DB2-groupname* is the name of the DB2 data sharing group.

The lock structure consists of two parts, which work completely independently of each other:

- ▶ *Lock table or hash table*

The lock table is where members register their interest in a resource. The lock table contains the lock status information and the owning member of those locks, and it is used to provide global lock serialization. The purpose of the lock table is to detect global lock contention. When a lock that is required by a member needs to be made known to other members in the group, it is registered in the lock table.

- ▶ *Modified resource list or record list table*

The IRLMs in the group use the modified resource list to store runtime information about the modify locks (X-type locks) owned by their DB2 members. The modified resource list records locks that protect changed data, thereby protecting the data in case of failure. If a lock is to be preserved across a subsystem failure, it must be stored in the modified resource list.

The following are some features provided when using the coupling facility lock structure:

- ▶ Fast detection of possible inter-DB2 conflicts.
- ▶ Two lock states supported: S and X.
- ▶ Lock conflict detected at hash entry level.
- ▶ No queuing is supported by the coupling facility architecture.

Figure 10-5 shows the two components of the lock structure.

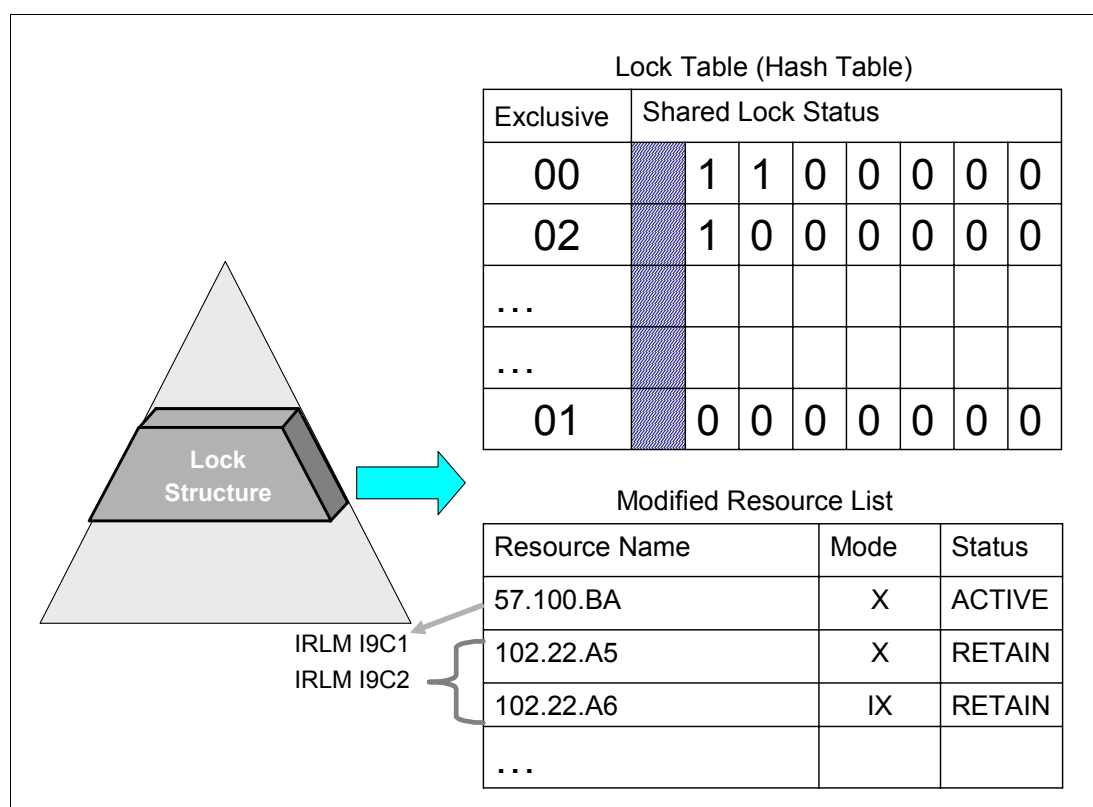


Figure 10-5 Coupling Facility lock structure

The lock table part of the lock structure is nothing more than a large hash table. It is composed of several (typically, at least a few million) lock table entries (or hash class or lock entries), each of them containing:

- ▶ Global byte, indicating the (IRLM ID with) *exclusive* interest in the hash class. This member owns the exclusive lock on the resource, and it is known as the global lock manager (GLM) for this resource.
- ▶ Share bits, one bit per (DB2) IRLM member, indicating *Share* interest in the hash class. In this case, there are seven share bits; the first bit is reserved.

The first hash entry in the lock table in Figure 10-5 shows that IRLM-ID 01 (I9C1) and IRLM-ID 02 (I9C2) have share interest (the shared lock status bit for both of the members is “1”) while no IRLM has an exclusive interest for this entry (the exclusive byte is “00”).

The second hash entry shows that IRLM I9C2 has an exclusive lock on the resource hashing to this entry while IRLM I9C1 has a share interest.

The record list contains all the currently active modify locks and retained locks for all members in the DB2 data sharing group. When a lock is used that allows an object to be changed (called a modify lock), the lock is kept in the record list of the coupling facility lock structure to allow for recovery in case a member fails. If a member is currently failed, then the record list entries for that member contain all of the retained locks for that member. Figure 10-5 shows that the record list active entry for the IRLM I9C1 and the two retained record list entries for the failing DB2 IRLM I9C2.

## Long duration locks

IRLM supports the concept of “long duration” and “short duration” locks. The idea is to try to separate those locks that are held for long periods of time into their own private section of the CF lock table so that they do not cause false contention with the shorter duration locks that more frequently come and go.

IRLM uses a small subset of the lock table entries for “long duration” locks. The remaining lock table entries are used for the “short duration” locks. Because “long duration” locks use only a small percentage of the available lock table entries and because the “long duration” locks typically hang around for a long period of time, they are more likely to suffer from false contention than “short duration” locks. See 10.10.3, “False contention” on page 354 for false contention details.

## 10.4 L-locks versus P-locks

The serialization techniques that are used to maintain the logical and physical consistency of inter-transaction shared data in a single system are not sufficient in a multi-system data sharing environment. They must be extended to operate across multiple systems. Specifically, the following two serialization techniques have to be addressed:

- IRLM locks: In a single DB2 system environment, IRLM maintains all DB2 locks in a local set of “software lock structures” inside the IRLM address space (most of it is in virtual storage above the 2 GB bar). In this configuration, all locks have a single-system scope, or local locks, that is, the effects of the lock are seen only within the single DB2 subsystem.

For data sharing, the scope of the IRLM locks must be extended from single-system to multi-system, that is, the effects of the lock must be visible not only to the local DB2 subsystem, but also to the other DB2 subsystems that are sharing access to the same databases across the DB2 data sharing group. Locks that have a multi-system scope are called global locks.

- DB2 latches: A latch is a memory-based serialization technique (as opposed to “lock”, which is name-based) to ensure physical consistency of shared DB2 resources (for example, pages in the buffer pool). Latches are faster than locks, and are generally held for shorter durations. In some cases when page locks are not used (for example, row level locking or space map page updates), DB2 relies on the “page latch” to ensure that only one transaction at a time is physically modifying the page.

In a data sharing environment, to provide the same level of inter-transaction physical serialization, we need to extend the scope of the page latch from single-system to multi-system. This is done by way of physical locking (P-locking) of the page, or page P-locks. Other types of DB2 latches are not affected by data sharing; they still exist and work exactly the same as in a non-data sharing environment.

In DB2 data sharing, the concurrency locks used by transactions are called *logical locks* or *L-locks*. These are the type of locks that also exist in a single DB2 system. They are used to control the concurrent access to objects and are normally associated with programs. L-locks are also referred as *transaction locks*. They can have deadlocks or timeouts and they are not negotiable. Refer to 3.3, “The Internal Resource Lock Manager (IRLM)” on page 44 for the different modes of L-Locks.

A DB2 data sharing environment also uses another kind of global lock, called *physical locks*, or *P-locks*. There are mainly two types of P-locks:

- ▶ The *page set/partition physical locks* are used to track inter-DB2 data coherency at the page set or partition level.
- ▶ The *page physical locks* are used to provide (write) serialization between members, ensuring the physical consistency of a page, just as latches do in a DB2 non-data sharing environment.

Other types of P-locks include DBD, castout, GBP structure, index tree, and repeatable read tracking P-locks.

**Note:** In this publication, we refer to page set/partition physical locks, which is the correct terminology, as page set P-lock for simplicity.

P-locks are not used for data concurrency control, but are used for data consistency control.

P-locks do not have deadlocks or timeouts. They are owned at the DB2 member level and they can be negotiated.

Table 10-5 compares the main distinguishing features of L-locks and P-locks.

Table 10-5 Distinguishing features between L-locks and P-locks

Category	L-locks	P-locks	
		Page set P-lock	Page P-locks
Used for	Concurrency	Coherency	Consistency
Owned by	Transaction or program	DB2 member	DB2 member
Duration	Manual, commit, and allocation	Interest	Update duration
Negotiable?	No	Yes	Yes
Timeout/deadlock	Yes	No	No
Stored where?	IRLM, XES, and CF	IRLM, XES, and CF	IRLM, XES, and CF

## 10.5 How inter-DB2 data coherency or read-write interest works

Page set or partition P-locks are used to track *inter-DB2 read/write interest (IDRWI)*, that is, to indicate the level of interest each DB2 member has in a open page set or partition. Each member acquires one (and only one) page set P-lock for each open table space or index, or for any partition of a partitioned table space or partitioned index. Page set or partition P-locks are owned by the member, they are independent of the transactions, and do not control concurrency.

Page set/partition P-locks are always global locks and the duration is determined by the interest of the DB2 member. A DB2 subsystem obtains a page set P-lock on a data set when it opens the data set, and holds the pageset P-lock until the data set is physically closed on that DB2 subsystem. A DB2 system will have a page set P-lock as long as it is “interested” in that page set, and therefore the page set P-lock can be held for many hours, days, or weeks.

Pageset P-locks are always propagated to the lock structure in the coupling facility at member level.

The concepts of inter-DB2 read/write interest on a page set or partition and group buffer pool dependency (*GBP-dependency*) are closely related.

Whenever inter-DB2 read/write interest exists on a page set or partition, that object is GBP-dependent. Conversely, if no inter-DB2 read/write interest exists on a page set or partition, the object is usually not GBP-dependent. Sometimes an object still has pages cached in the group buffer pool, and it can remain GBP-dependent even after the inter-DB2 read/write interest has gone away. For more information about GBP-dependency on caching data into group buffer pool, refer to *DB2 9 Data Sharing Planning and Administration Guide*, SC18-9845.

The inter-DB2 interest of a page set physical lock for a given page set or partition by a given member is determined by the following activities on the page set or partition:

- ▶ Physical open, read-only interest (none → RO)
- ▶ Physical close, no interest (any → none)
- ▶ First update or pseudo-open, read-write interest (RO → RW)
- ▶ RO-switching or pseudo-close, read-only interest (RW → RO)

A member's interest goes from none to RO at physical open, RO to RW at pseudo-open, RW to RO at pseudo-close, and any to none at physical close.

Page sets go in and out of GBP-dependency using the page set P-locking mechanism (page set P-lock negotiation using the P-lock exit). In this section, we explain the page set P-lock negotiation scheme and the relationship with inter-DB2 interest. For more information about getting in and out of GBP-dependency, refer to 10.8, "Pseudo-close and the GBP-dependency" on page 334.

Page set P-lock negotiation occurs when one of the DB2 members changes the state of its P-lock on a resource (page set or partition) due to a change in the physical access characteristics on the resource (for example, the DB2 member's interest changes from read only ("RO") to read/write ("R/W") on a particular page set). The other DB2 members that hold a page set P-lock on that resource will be notified by their respective IRLMs of this change in the inter-DB2 interest on the P-lock, and each DB2 member can then dynamically make the necessary adjustments in the cache coherency processing for the resource and then downgrade or upgrade its P-lock state (negotiate the P-lock) accordingly. The page set P-lock negotiation process allows DB2 members to react dynamically to the changes of inter-DB2 interest and to enact intersystem buffer coherency protocols only when there is actual physical inter-DB2 R/W sharing of a page set. P-lock negotiation between each member's IRLM are done through XCF messaging.

Table 10-6 shows the page set P-Lock lock modes depending on the DB2 system holder interest and whether a page set or partition is GBP-dependant:

Table 10-6 Page set P-lock modes depending on the DB2 member interest

Interest of the requesting DB2	Interest of other members	Page set P-lock mode of requesting DB2	GBP-dependent
Read-only	None or read-only	Shared (S)	No
Read-only	Read-write	Intent Share (IS)	Yes
Read-write	None	Exclusive (X)	No
Read-write	Read-only	Share with Intent Exclusive (SIX)	Yes
Read-write	Read-write	Intent Exclusive (IX)	Yes

For example, when a the DB2 member is going from RO to R/W interest, or pseudo-open on a page set/partition, and the other DB2's mode of interest is RO, the P-lock mode on that page set or partition will be negotiated and end up with a lock mode of SIX. The DB2 members that have the RO interest will have the P-lock mode of IS. The page set or partition will be GBP-dependent.

### 10.5.1 Inter-DB2 read/read interest

To illustrate the process in more detail, we use Figure 10-6. It describes a situation where two DB2 members are readers, that is, *inter-DB2 Read/Read Interest (IDRRI)*. On each member there is a transaction reading data from table space A (Tbs A). Tbs A will be open for read on each member. Using Table 10-6, this means that the page set P-lock mode for Tbs A will be a shared P-lock (S).

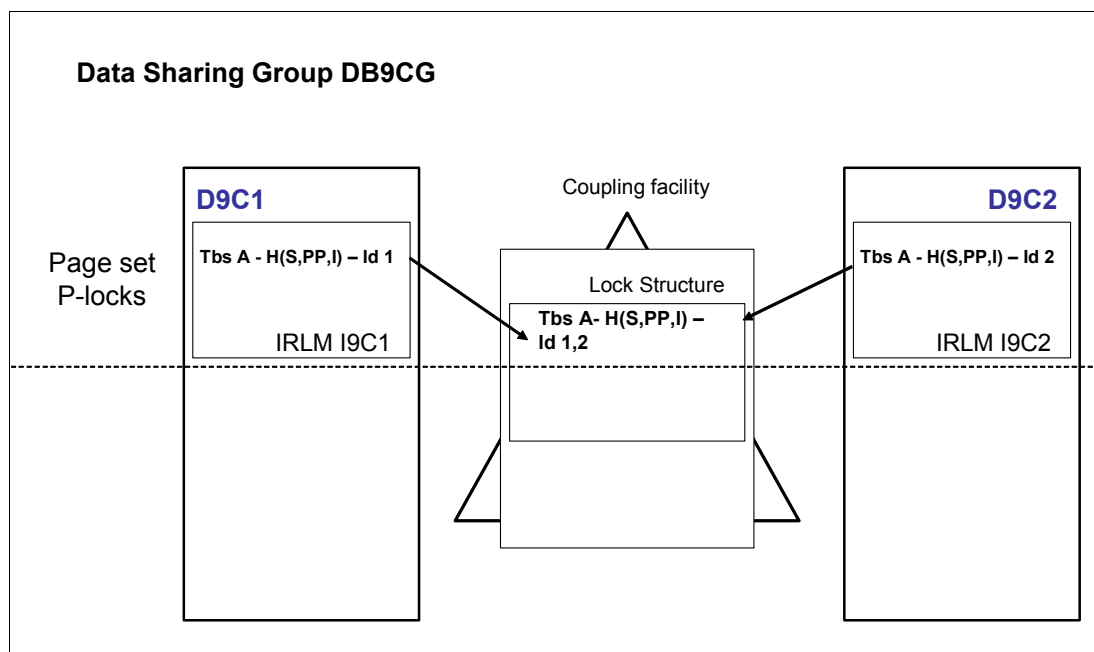


Figure 10-6 Data sharing page set/partition P-lock: two readers

The nomenclature used is Figure 10-6 on page 317 and the following figures is as follows:

Resource - H(S,PP,I) - Member ID

Where:

<b>Resource</b>	Table space or partition
<b>H</b>	Held ("H" means that this lock is held.)
<b>S</b>	Share (This lock is share level.)
<b>PP</b>	P-lock (This is a page set physical lock.)
<b>I</b>	Interest (This lock is held by a DB2 system for as long it is interested in the page set.)

These conventions are somewhat similar to the LOCKINFO output that is part of the -DIS DATABASE LOCKS command. Note that the nomenclature used for the coupling facility lock structure is only intended to make things easier to understand; this nomenclature does not show up in any command output.

Refer to 9.1.1, "DB2 commands" on page 256 for LOCKINFO output details.

In Figure 10-6 on page 317, DB2 member D9C1 holds an S (shared) pageset P-lock for as long as a read interest in the page set (A - H(S,PP,I) - 1) exists. DB2 member D9C2 has the same interest (A - H(S,PP,I) - 2). The existence of two S level pageset P-locks indicates that there are no updaters on either of the DB2 members. Pageset P-locks are always propagated to XES and sent to the coupling facility. In the coupling facility lock structure, the S shared level for page set or partition P-lock are registered for both members (A - H(S,PP,I) - 1,2).

The -DISPLAY DATABASE LOCKS command output in Example 10-1 shows the information regarding the page set/partition P-lock for our both readers.

*Example 10-1 DISPLAY DATABASE ... LOCKS command: two readers*

```

***** Top of Data *****
DSNT360I  -D9C2 *****
DSNT361I  -D9C2 *  DISPLAY DATABASE SUMMARY
              *  GLOBAL LOCKS
DSNT360I  -D9C2 *****
DSNT362I  -D9C2  DATABASE = RITA1229 STATUS = RW
              DBD LENGTH = 28256

DSNT397I  -D9C2
NAME      TYPE PART  STATUS              CONNID  CORRID  LOCKINFO
-----
GLWSPRJ  TS        RW                  TSO      DB2S2   H-IS,S,C
-
-        AGENT TOKEN 2628
-        MEMBER NAME D9C1
GLWSPRJ  TS        RW                  TSO      DB2S2   H-S,PP,I
-
-        MEMBER NAME D9C1
GLWSPRJ  TS        RW                  TSO      DB2S2   H-S,PP,I
-
-        MEMBER NAME D9C2
GLWSPRJ  TS        RW                  TSO      DB2R2   H-IS,S,C
-
-        AGENT TOKEN 1037
-        MEMBER NAME D9C2
31       TB        RW                  TSO      DB2S2   H-IS,T,C
-
-        AGENT TOKEN 2628
-        MEMBER NAME D9C1

```



```

31      TB                      TSO      DB2R2      H-IS,T,C
      -      AGENT TOKEN 1037
      -      MEMBER NAME D9C2
***** DISPLAY OF DATABASE RITA1229 ENDED *****
DSN9022I  -D9C2 DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION

```

Table space GLWPRJ (which is segmented) is being accessed by two correlation IDs, DB2S2 and DB2R2, in members D9C1 and D9C2, respectively. The two DB2 members hold the pageset P-lock (PP) in S mode, for interest duration on the table space. Page set P-locks are identified by a member name rather than a correlation ID (the CORRID field is blank in the command output). The output also contains the L-locks that have been acquired. We discuss these later in this section.

## 10.5.2 Inter-DB2 read/write interest

The next example (Figure 10-7) presents a scenario where the two DB2 members are in an inter-DB2 read/write interest (IDRWI) situation, that is, one reader and one updater. DB2 member D9C1 is the reader and holds an IS page set P-lock on the page set A (Tbs A - H(IS,PP,I) - Id 1).

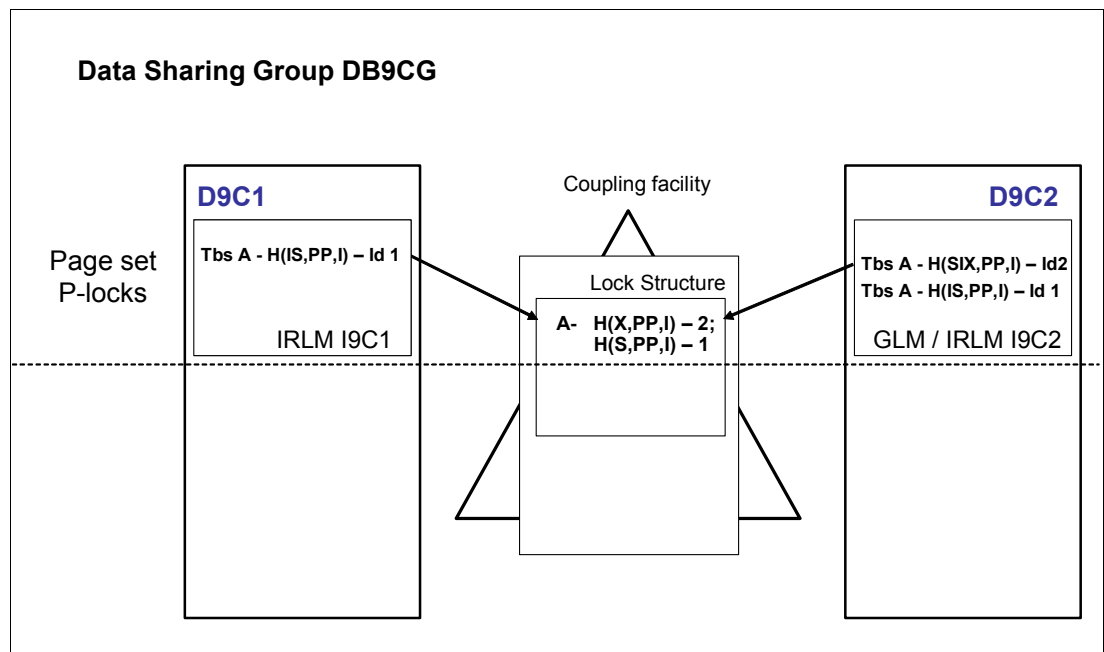


Figure 10-7 Data sharing page set/partition P-lock: inter-DB2 read/write interest

When the first update transaction that accesses Tbs A arrives on member D9C2, D9C2 requests an X page set P-lock, and member D9C1 is holding a S page set P-lock at that point, as it has table space A open for read. An X and an S lock are not compatible, but because P-locks can be negotiated, member D9C2 does not have to wait for member D9C1 to give up its S P-lock. The two DB2s negotiate the P-lock down to a level where we again have two compatible page set P-locks. In this case, the reader, D9C1, downgrades from S to IS (Tbs A - H(IS,PP,I) - Id 1), and the updater, D9C2, downgrades from his initial request for an X lock to a SIX page set P-lock (Tbs A - H(SIX,PP,I) - Id 2). Since IS and SIX are compatible, page set A can be shared by both members. This process is known as *page set P-lock negotiation*.

In an inter-DB2 read/write interest (or inter-DB2 write/write interest) situation, when a page set P-lock conflict arises, it is resolved through page set or partition P-lock negotiation.

The SIX page set P-lock is propagated to the lock structure in the CF as an XES X-lock (A - H(X,PP,I) - 2) and the IS page set P-lock is propagated as (A - H(S,PP,I) - 1), as XES only knows S and X type locks. Assuming that no other resources hash to the same lock table entry, the XES component for D9C2 now becomes the global lock manager (GLM) for this lock table entry in the lock structure. Whenever a new request comes in for a lock table entry, the GLM is responsible for handling conflicts, and negotiations of the lock. Not only XES, but also DB2 member D9C2's IRLM I9C2, will have all the information about the other DB2 member's interest in the page set. Please note that the information kept by IRLM is for serviceability only. The Global management ownership is decided by XES. The GLM information kept in IRLM plays no role in making any global resource management decision. Refer to 10.10.1, "Global lock manager" on page 353 for more information about the global management process.

The I9C2 IRLM (that is associated with the XES global lock manager) knows each of the page set P-lock modes (IS and SIX) that X mode XES lock for the page set in the coupling facility represents. Whenever a new incoming request global lock is made for page set A, the global lock manager XES cannot resolve the contention, so control is passed back to D9C2's IRLM. IRLM I9C2 then sorts out the contention and, if possible, grants a compatible lock after communicating through XCF with IRLM I9C1.

Example 10-2 shows the output for the -DISPLAY DATABASE LOCKS command for a case with inter-DB2 read/write interest. The LOCKINFO field shows H(IS,PP,I) for member D9C1 (the reader) and H(SIX,PP,I) for member D9C2 (the updater).

At this stage, the object (GLWSPRJ) is group buffer pool dependent.

*Example 10-2 -DISPLAY DATABASE LOCKS command: inter-DB2 read/write interest*

---

```

DSNT360I  -D9C1 *****
DSNT361I  -D9C1 *   DISPLAY DATABASE SUMMARY
              *   GLOBAL LOCKS
DSNT360I  -D9C1 *****
DSNT362I  -D9C1      DATABASE = RITA1229 STATUS = RW
              DBD LENGTH = 28256

DSNT397I  -D9C1
NAME      TYPE PART  STATUS          CONNID   CORRID   LOCKINFO
-----
GLWSPRJ   TS        RW              TSO      DB2R2    H-IX,S,C
-          -          AGENT TOKEN 1042
-          -          MEMBER NAME D9C2
GLWSPRJ   TS        RW              (CO)1      H-SIX,PP,I
-          -          MEMBER NAME D9C2
GLWSPRJ   TS        RW              H-IS,PP,I
-          -          MEMBER NAME D9C1
GLWSPRJ   TS        RW              TSO      DB2S2    H-IS,S,C
-          -          AGENT TOKEN 2644
-          -          MEMBER NAME D9C1
31        TB        TSO      DB2R2    H-IX,T,C
-          -          AGENT TOKEN 1042
-          -          MEMBER NAME D9C2
31        TB        TSO      DB2S2    H-IS,T,C
-          -          AGENT TOKEN 2644
-          -          MEMBER NAME D9C1

```

---

<sup>1</sup> The characters '(CO)' following a member name indicate that this member is the castout owner for the page set or partition. This DB2 member is responsible for doing all of the castout I/O for that page set or partition.

### 10.5.3 Inter-DB2 write/write interest

A inter-DB2 write/write interest scenario is shown in Figure 10-8. Both DB2 members D9C1 and D9C2 are running transactions that are updating table space A. Each member holds an IX page set P-lock (refer to Table 10-6 on page 317). In this example, DB2 member D9C1 is now the global lock manager. Initially, D9C1 held an X page set P-lock and when D9C2 requested an X mode page set P-lock, negotiation took place. As two X page set P-locks are incompatible, both members downgrade to IX ( $H(IX,PP,I)$ ). The coupling facility lock structure has an X page set P-lock ( $H(X,PP,I)$ ) for D9C1 (Id 1), as this member is the global lock manager for the resource. For a lock request that generates contention, the global lock manager will resolve or confirm contentions on behalf of other resources hashing to the same lock table entry.

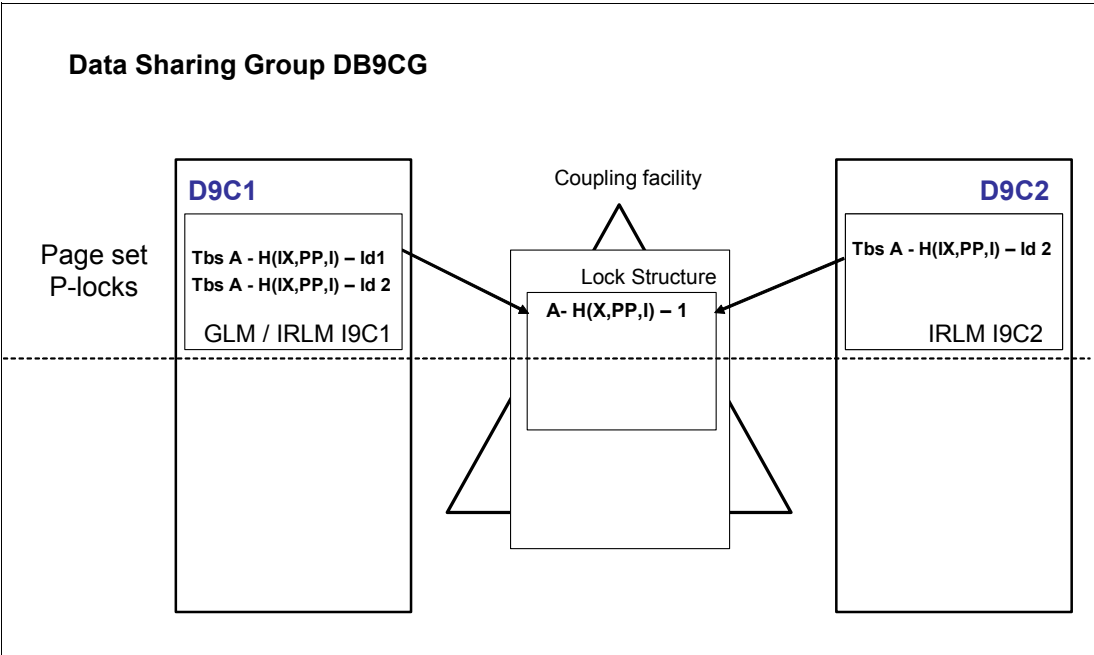


Figure 10-8 Data sharing page set/partition P-lock: inter-DB2 write/write interest

The -DISPLAY DATABASE LOCKS command output in Example 10-3 shows the page set/partition P-locks when there are multiple members that have transactions that are updating the same resource (page set).

*Example 10-3 -DISPLAY DATABASE LOCKS command: inter-DB2 write/write interest*

---

```

DSNT360I  -D9C1 *****
DSNT361I  -D9C1 *   DISPLAY DATABASE SUMMARY
              *   GLOBAL LOCKS
DSNT360I  -D9C1 *****
DSNT362I  -D9C1      DATABASE = RITA1229 STATUS = RW
              DBD LENGTH = 28256
DSNT397I  -D9C1
NAME      TYPE PART  STATUS              CONNID   CORRID   LOCKINFO
-----
...
GLWSPRJ   TS        RW                    H-IX, PP, I
-          MEMBER NAME D9C2
GLWSPRJ   TS        RW                    H-IX, PP, I
-          MEMBER NAME D9C1      (CO)
...
***** DISPLAY OF DATABASE RITA1229  ENDED *****

```

---

Note that the object (GLWSPRJ) is group buffer pool dependent and that D9C1 is the castout owner (CO) for this resource. This does not necessarily mean that the global lock manager will be the IRLM/XES that is associated with D9C1. Global lock management and castout ownership are managed completely independent of each other.

In summary, page set or partition P-locks do not serialize access to a resource. They are used to track which members have interest in a resource, and determine when a resource must become GBP-dependent.

Page set P-locks are released when a page set or partition data set is closed. When page set P-locks are downgraded (or released), GBP-dependency is reevaluated.

## 10.6 Data sharing locking optimizations

If every global lock request were to be propagated beyond the local IRLMs, there would be a significant performance degradation to the system. Therefore, one of the major design goals of the locking scheme in DB2 data sharing was to minimize the number of times a “really global” lock request must be processed. Data sharing uses some important global locking optimization techniques to reduce the necessity of processing L-locks beyond the local IRLM whenever possible:

- ▶ When no inter-DB2 read/write interest exists for an object, explicit hierarchical locking avoids processing a number of locks beyond the local IRLM.
- ▶ If a single member with update interest and multiple members with read-only interest exist, IRLM propagates fewer locks than when all members have update interest in the same page set.
- ▶ All P-locks that are held beyond the local IRLM are owned by a member, not by an individual work unit. This reduces lock propagation by requiring that only the most restrictive lock mode for an object on a given member be propagated to the coupling facility. A new lock that is equal to, or less restrictive than the lock currently being held, is not propagated.

- IRLM can release many locks with just one request to the coupling facility. This can occur, for example, after a transaction commits and has two or more locks that need to be unlocked. It also can occur during DB2 shutdown or abnormal termination when the member has two or more locks that need to be unlocked.

### 10.6.1 Lock optimization example

The lock optimization example presented here emphasizes the point that XES only tracks lock ownership at the DB2 member level. XES does not map lock ownership down to the transaction level as IRLM does. This fact allows for an important global locking optimization, that is, for a given resource, IRLM only needs to propagate the most restrictive state to XES.

If a lock is requested on a resource being accessed by another transaction on the local DB2 member that already holds a lock on the resource in an equal or more restrictive (but compatible) state, then the lock can be granted locally by IRLM. This can be very important to reduce global locking costs for table space and partition intent locking (IX and IS locks) in a high multi-programming environment. This can also come into play for S and U mode page (or row) locks that are held and requested concurrently on the same resource by multiple transactions.

This effects for the page locks is less noticeable than for the table space and partition locks, because the page locks are typically held for much shorter durations and have a much finer granularity, and thus are less likely to have high levels of concurrent access.

If IRLM determines that it needs to propagate the global lock to XES, it does so only after it has determined that the lock is locally grantable (when there is no inter-transaction contention for the lock within the local DB2 member).

Figure 10-9 shows an example of this situation. Events occur in time sequence order.

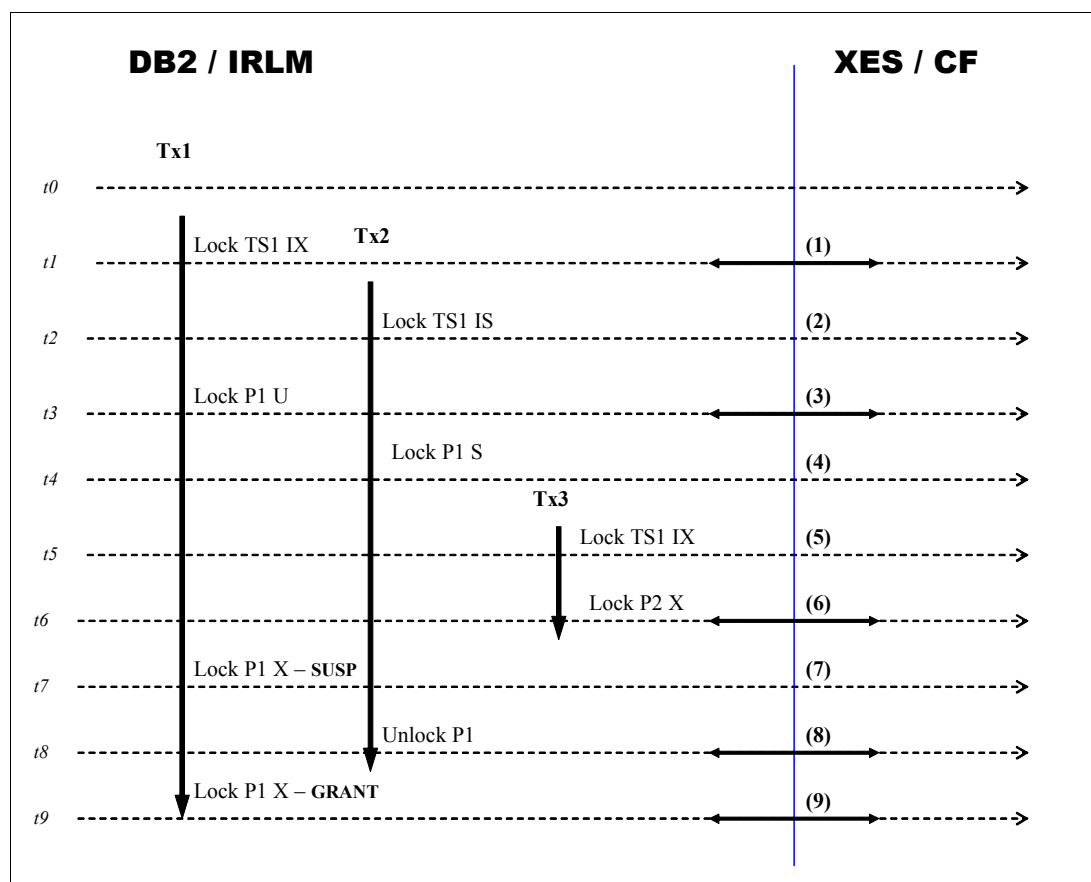


Figure 10-9 Lock optimization example

The purpose of this example is to show that:

- ▶ IRLM only propagates the most restrictive state per resource to XES.
- ▶ IRLM only issues a request to XES after the lock is locally grantable.
- ▶ The optimization is also applicable to S and U locks.

The XES/CF access requests are shown on the right side of the figure and are described as follows

1. Transaction Tx1 requests an IX lock on table space TS1. DB2 sends the request to IRLM. IRLM propagates the request to XES, and the lock is granted synchronously (no suspend).
2. Tx2 requests an IS lock on table space TS1. DB2 sends the request to IRLM. IRLM determines that the lock is locally grantable (there are no incompatible holders or higher priority waiters). IRLM sees that an IX lock has already been propagated to XES for this resource, so there is no need to propagate this IS lock, and the lock is granted locally by IRLM.
3. Tx1 requests a U lock on page P1. DB2 sends the request to IRLM. IRLM propagates the request to XES, and the lock is granted synchronously (no suspend).
4. Tx2 requests an S lock on page P1. DB2 sends the request to IRLM. IRLM determines that the lock is locally grantable. IRLM sees that a U lock has already been propagated to XES for this resource, so there is no need to propagate this S lock, and the lock is granted locally by IRLM.

5. Tx3 requests an IX lock on table space TS1. DB2 sends the request to IRLM. IRLM determines that the lock is locally grantable. IRLM sees that an IX lock has already been propagated to XES for this resource, so there is no need to propagate this IX lock, and the lock is granted locally by IRLM.
6. Tx3 requests a X lock on page P2. DB2 sends the request to IRLM. IRLM propagates the request to XES, and the lock is granted synchronously (no suspend).
7. Tx1 tries to upgrade its lock on page P1 from U to X. DB2 sends the request to IRLM. But IRLM detects local contention on P1 (Tx2 still holds an S lock on P1), so Tx1 is suspended. XES is not invoked.
8. Tx2 releases its S lock on P1. DB2 sends the unlock request to IRLM. IRLM unlocks the lock and sees that Tx1's X request is now grantable. Because the most restrictive lock state on this resource is being upgraded from U to X, IRLM must invoke XES to notify XES of the upgrade. IRLM sends the request to XES and returns control to Tx2.
9. The request to notify XES of the U to X upgrade returns successfully and Tx1 is resumed and granted its X lock on P1.

Once IRLM knows that a resource is in global management, it propagates all subsequent requests for that resource to XES.

## 10.6.2 Explicit hierarchical locking

Explicit hierarchical locking allows IRLM to grant “child” L-locks locally when no inter-DB2 read/write interest exists on the parent. Granting L-lock requests locally, versus globally, improves performance. In this case, only the “parent” L-locks are propagated. To make explicit hierarchy locking work, DB2 resources are organized using an explicit parent/child hierarchy, as shown in Figure 10-10.

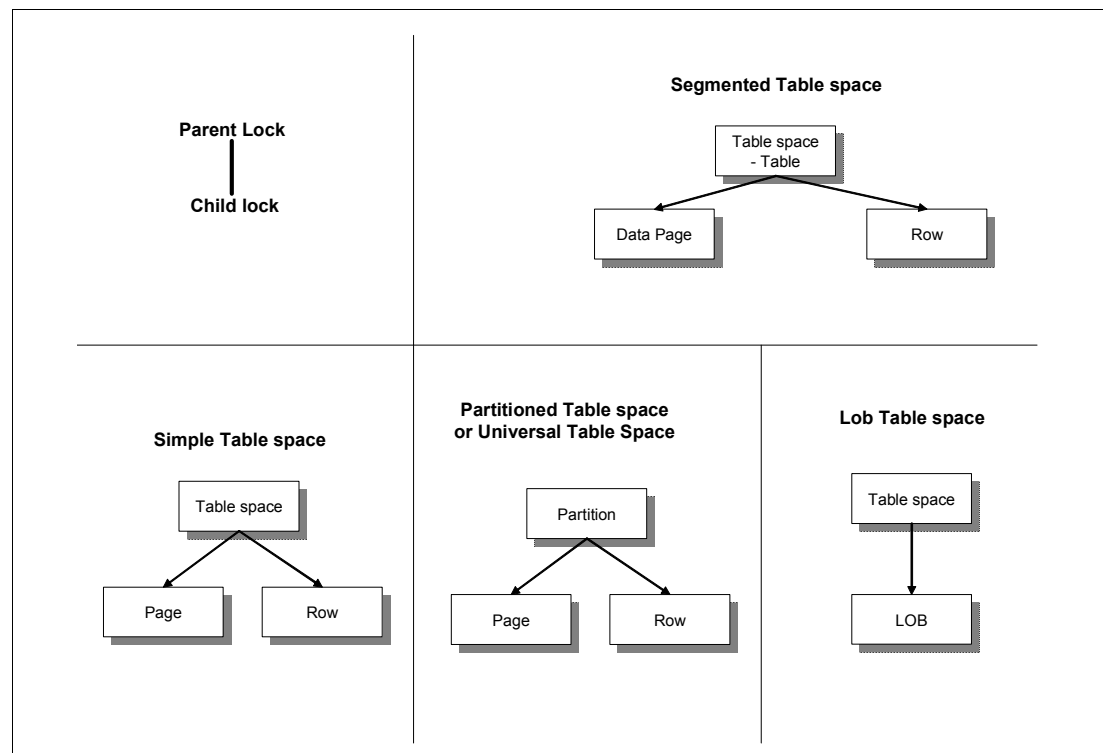


Figure 10-10 Explicit parent/child hierarchy

Within IRLM, a hierarchy exists between certain types of L-locks, where a parent L-lock is the lock on a page set (table space or partition) and a child L-lock is the lock held on either the table, data page, or row within that page set. For partitioned or universal table spaces, each locked partition is a parent of the page or row child L-locks held for that partition.

## Parent L-lock propagation

In a DB2 data sharing environment, the first parent L-lock on a resource is always propagated to XES and the coupling facility lock structure. If a parent L-lock has already been propagated to XES, protecting a particular resource for this member, subsequent lock requests for the same lock do not have to be sent to XES by the same member for the same resource. They can be serviced locally.

In addition, a parent L-lock is only propagated if it is more restrictive than the current status that XES knows about for this resource from this member. This reduces lock propagation by requiring that only the most restrictive lock mode for an object on a given member has to be propagated to XES and the coupling facility. For example, if an IX L-lock has already been propagated to XES for this member, a subsequent IX L-lock does not have to be sent.

As for single system environments, the value of the RELEASE parameter, which is specified at bind time, determines whether parent L-locks are released when the transaction commits — RELEASE(COMMIT), or when the thread terminates — RELEASE(DEALLOCATE). For more information regarding the RELEASE BIND option, refer to 12.2.2, “Table space lock duration” on page 427.

Child L-locks (table, page, or row) are propagated depending on the compatibility of the page set P-lock on this member with the page set P-locks that are held by other members for the page set. Since DB2 for z/OS V8 with locking protocol level 2, the inter-DB2 interest on the page set P-lock controls both GBP-dependency and child lock propagation.

Table 10-7 shows when and which child locks are being propagated to XES depending on the page set or partition P-lock mode.

*Table 10-7 Determining when child locks are propagated to XES*

Mode of interest of this member	Maximum page set P-lock mode of the other members	Are X L-child locks propagated by this member?	Are S L-child locks propagated by this member?	Are U L-child locks propagated by this member?
IS, S	NONE, IS, S	N/A	No	Yes
X	NONE	No	No	No
IS	IX, SIX	N/A	Yes	Yes
IX, SIX	IS	Yes	No	Yes
IX	IX	Yes	Yes	Yes
SIX	NONE	Yes	No	No
IX	NONE	No	No	No

As shown in Table 10-7, U child L-locks do not force an upgrade of the pageset P-lock, but they can be propagated in order to ensure that U is incompatible with U on another member.



Figure 10-11 continues the scenario we started in 10.5.1, “Inter-DB2 read/read interest” on page 317, extending it with parent L-locks.

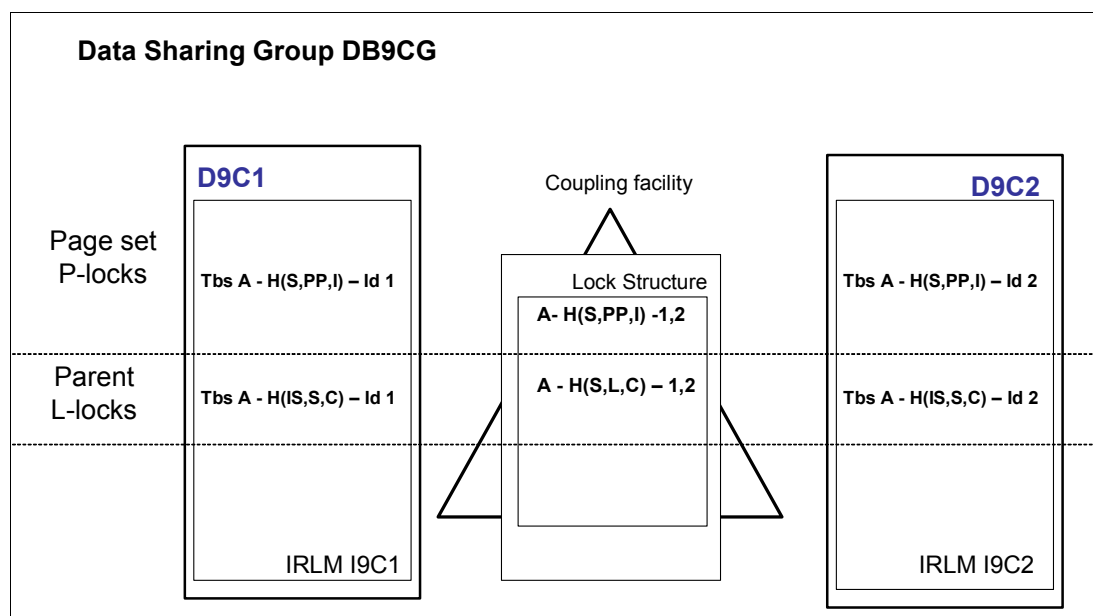


Figure 10-11 Data sharing parent L-Lock: inter-DB2 read-read interest

The nomenclature now used is as follows:

Resource - H(IS,S/L,C) - Member ID

Where:

<b>Resource</b>	Table space, table, or page.
<b>H</b>	Held (“H” means that this lock is held).
<b>IS</b>	Intent to Share (a lock with read intentions).
<b>S</b>	A table space L-lock.
<b>L</b>	L-lock (this is a logical lock).
<b>C</b>	The lock is released at commit.

The reader transactions in both D9C1 and D9C2 members are accessing the same table space. Figure 10-11 shows the propagation of the parent L-locks based on the state of the page set P-lock. Each DB2 member propagates its parent L-lock to XES and the coupling facility lock structure (as parent L-locks are always propagated to the coupling facility lock structure), but no child locks are propagated, as no inter-DB2 read/write interest exists. D9C1 and D9C2 (Id 1 and Id 2) members need to propagate their intent to share (IS) L-lock on table space A(S) - H(IS,S,C). The L-lock in the parent resource hierarchy will be held until commit point (C). Based on the information in Table 10-2 on page 310, the IS L-locks are sent to XES as S-locks, so in the CF lock structure the lock state for table space A is an S “Share” parent L-lock for both DB2 members (A - H(S,L,C) - 1,2)).

The -DISPLAY DATABASE LOCKS command output from Example 10-1 on page 318 shows the parent L-locks on the table space. In our example, we used a segmented table space, so you can also see the L-locks for the table resource. Note that the -DISPLAY DATABASE LOCKS command does not allow you to tell whether locks were propagated to XES or not; only a trace allows you to determine whether or not that is the case.

## Selective partitions locking

When using a partitioned table space, parent locks are always obtained at the partition level. Individual partitions are locked as they are accessed. This locking behavior enables greater concurrency.

Since DB2 V8, pageset level locks are always acquired at the partition level for partitioned objects, even if the table space was defined in a version of DB2 prior to Version 8 using the LOCKPART NO clause. The LOCKPART clause is ignored.

Each locked partition is a separate parent lock. Therefore, DB2 and IRLM can detect whether or not inter-DB2 read/write interest exists at the partition level and does not propagate child L-locks unnecessarily on a partition.

Note that if any of the following conditions are true, DB2 cannot selectively lock the partitions. In this case, DB2 must acquire a lock on all the partitions, which may increase the impact of DB2 data sharing global locking:

- ▶ The plan is bound with ACQUIRE(ALLOCATE).
- ▶ The table space is defined with LOCKSIZE TABLESPACE.
- ▶ LOCK TABLE IN EXCLUSIVE MODE is used (without the PART option).

However, none of these conditions should be considered common practice.

## Child L-lock propagation

Now we continue our inter-DB2 read/write example that we started in Figure 10-7 on page 319. DB2 member D9C1 has read interest in page set A and D9C2 has update interest for page set A, and add the parent and child L-locks to the picture. The result is shown in Figure 10-12.

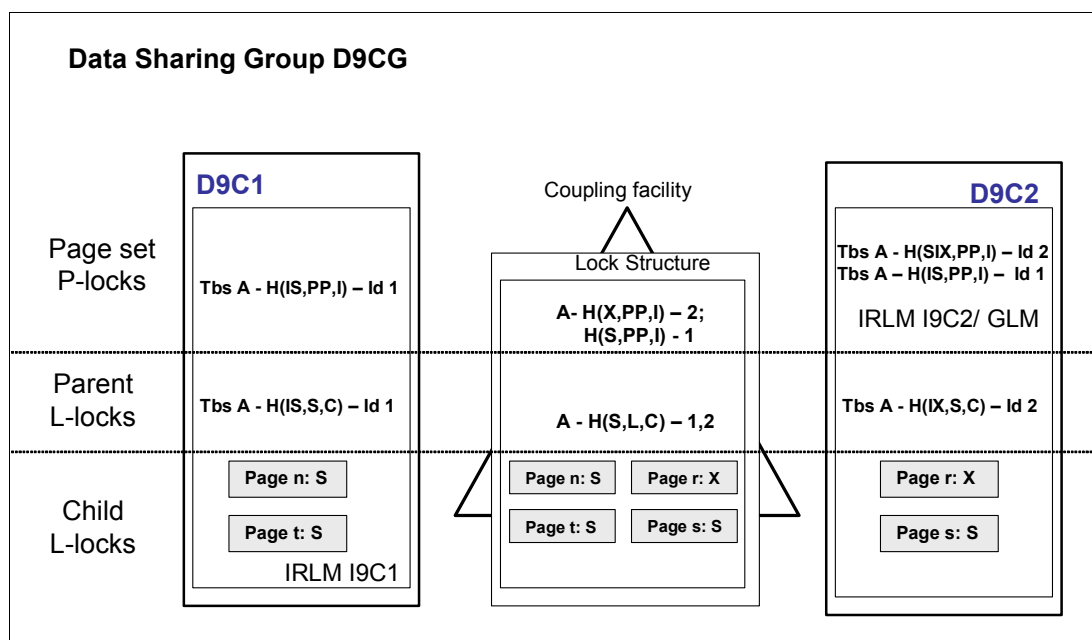


Figure 10-12 L-locks with inter-DB2 read/write interest

The reader transaction Tx1 on D9C1 member is reading data from page Pn and page Pt. The page set P-locks and parent L-locks acquired by the Tx1 are propagated to XES and the coupling facility lock structure. D9C1 propagated an IS page set P-Lock and IS table space L-lock on table space A (H(IS,PP,I) and H(IS,S,C)).

Transaction Tx2 on member D9C2 is reading page Pr and updating page Ps, which are different pages than the ones Tx1 is using. D9C2 also propagates its page set P-lock and the parent L-Lock (H(SIX,PP,I) and H(IX,S,C)).

The lock table entry is owned by XES on D9C2, which is the GLM for that entry (H(SIX,PP,I) - 2). The lock table entry for the parent L-lock in the coupling facility lock structure is not “owned” by D9C2 as it is for the pageset P-lock, because both parent L-locks (IS and IX) are propagated to XES as share mode locks, so there is no GLM for this lock table entry at this stage.

Using the child lock propagation information in Table 10-7 on page 326, IRLM I9C1 (of DB2 member D9C1) propagates all pages read (S mode child L-locks on pages Pn and Pt) by the reader transaction Tx1, and IRLM I9C2 propagates all child L-locks (S, U and X) acquired on pages (Pr and Ps) by the Tx2 updater transaction.

Indexes are not included in the hierarchy because index pages are protected by locks on the corresponding data.

## 10.7 Page P-locks

Page level physical locks (P-locks) are used to maintain physical page coherency in cases when a page can be updated by multiple DB2 members and inter-transaction subpage concurrency is allowed. Page P-locks extends the DB2 local page latch to “global scope”. You can think of a page P-lock as a multi-system extension of the page latch for a data sharing environment, and the page P-lock’s main function is to serialize updates on the same page at the same time by multiple DB2 members, that is, when inter-DB2 write/write exists.

For example, page P-locks are used when two subsystems want to update the same data page and row level locking is in effect. Page P-locks are also used for the updating space map pages and index leaf pages for GBP-dependent objects, regardless of the LOCKSIZE that is used.

The lock mode or state for page P-lock can be S share and X exclusive. The examples for X mode page P-locks are then:

- ▶ Update of a data page, when row locking is in effect for a GBP-dependent object
- ▶ Update of a space map page from a GBP-dependent object
- ▶ Update of an index leaf page from a of a GBP-dependent object
- ▶ Update of certain DB2 directory pages

S mode page P-locks are only required with:

- ▶ Repeatable read (RR) index scan with LOCKSIZE ROW for data pages and index leaf pages
- ▶ Scans of space map page for data or index, when inconsistencies are found
- ▶ Referential integrity (RI) checking

Like page set P-locks, page P-locks are owned by a DB2 member and are negotiable. The process of notifying another system to release the page P-lock (page P-lock negotiation) is done through a P-lock exit. The impact of frequent page P-lock negotiation can be high and expensive. Frequent page P-lock negotiation can have a significant impact on transaction response time and can also affect the overall system throughput.

Page P-locks can also be retained if a member fails.

Figure 10-13 illustrates the usage of page P-locks by DB2 when row level locking is used. It uses the following naming conventions:

Where:

<b>Resource</b>	Page or row.
<b>H</b>	Held (“H” means that this lock is held).
<b>pP</b>	Page P-Lock.
<b>X</b>	Mode of lock.
<b>L</b>	L-lock (this is a logical lock).
<b>C</b>	The lock is freed at commit.

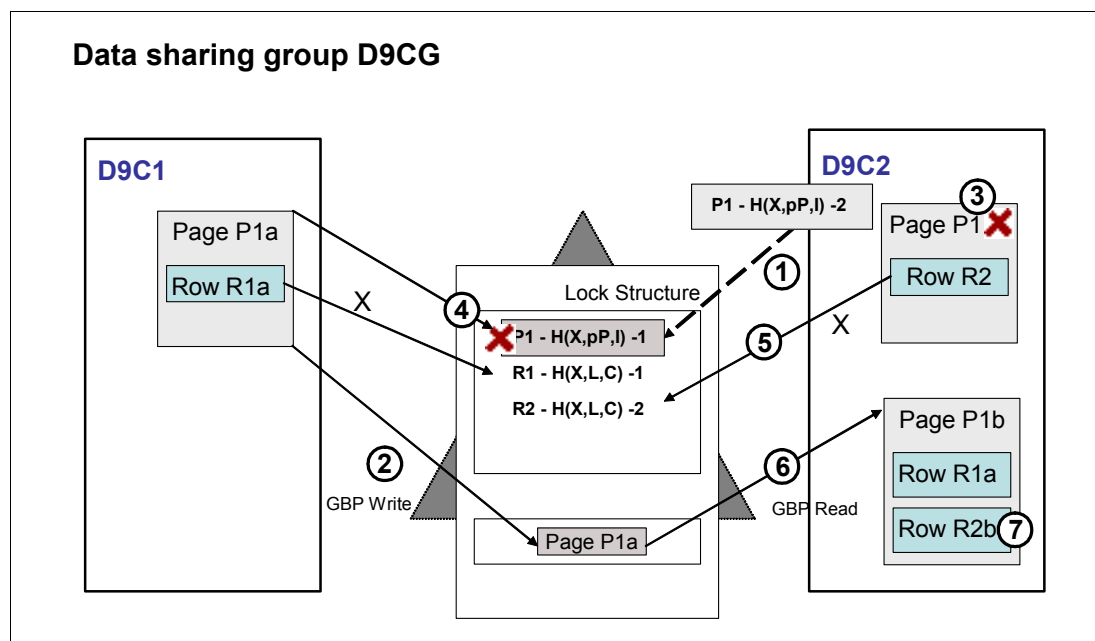


Figure 10-13 Page P-lock usage with LOCKSIZE ROW

Figure 10-13 on page 330 starts out where the page set is GBP-dependent (where D9C1 and D9C2 have an IX pageset P-lock (not shown)), and D9C1 wants to update row R1 on page P1. To achieve these goals:

- ▶ D9C1 obtains an X mode page P-lock to ensure the integrity of the page.
- ▶ The page P-lock is propagated to the coupling facility ( $P1 - H(X, pP, I) - 1$ )
- ▶ D9C1 obtains an X mode L-lock for row R1.
- ▶ The row L-lock is also propagated to the coupling facility ( $R1 - H(X, L, C) - 1$ ).
- ▶ D9C1 changes the row on page P1 to R1a and creates page P1a.

Now D9C2 needs page P1 to update row R2 on the same page P1. D9C1 has not committed its update yet. Here are the steps to provide data integrity:

1. D9C2 requests an X mode P-lock on page P1. That lock cannot be granted, as it is already held by D9C1, so the IRLMs of the two members negotiate for the P-lock, and D9C1 has to give up its page P-lock on page P.
2. Before D9C1 can give up its page P-lock, D9C1 has to force log, and write the page P1a to GBP (or disk (only for GBPCACHE NONE objects)).
3. When D9C1 writes the changed page P1a to the GBP, the local copy of page P1 in D9C2's local buffer pool will be invalidated.
4. Now that the latest copy of page P1a is written to the GBP, D9C1 can give up its X page P-lock. Note that D9C1 keeps the X mode L-lock on row R1, but D9C2 now owns the X mode page P-lock on page P1 ( $P1 - H(X, pP, I) - 2$ ).
5. D9C2 obtains the X mode L-lock for row R2 ( $R2 - H(X, L, C) - 2$ ).
6. As Page 1 has been cross invalidated in D9C2's local BP, page P1a is read in from the GBP.
7. D9C2 can now update row R2 to row R2a and creates page P1b.
8. No later than at commit, D9C2 writes the updated page P1b to the GBP.
9. This write causes D9C1's version of the page P1a to be invalidated. If D9C1 issues a commit (remember, D9C2 got the page before D9C1 had committed its changes to row R1a), it does not need to write out the page.

This is what happens when things go according to plan. However, what happens if D9C1 performs a rollback? It has to renegotiate the page P-lock so that it can change the row from R1a back to R1. Here is what will happen (not shown in Figure 10-13 on page 330):

1. D9C1 negotiates for the page P-lock.
2. D9C2 gives up the X mode page P-lock on page P1b (writing page P1b to the GBP as part of the process, and invalidates the version in the local BP of D9C1).
3. D9C1 acquires the X mode page P-lock on page P1b.
4. D9C1 reads page P1b in from the GBP. D9C1 rolls back the row to R1 on page P1b and creates page P1c.
5. No later than commit, D9C1 writes page P1c to the GBP.
6. The page in D9C2's local buffer pool is invalidated.
7. D9C1 releases its L-lock on row R1 at commit.

As you can see from this scenario, while all logical locking is done at the row level, the physical locking is done at the page level. This type of page P-locks usage and propagation (with the GBP pages cross invalidation) is only performed when row level locking is in effect.

However, this does not mean that with page level locking that there are no page P-locks. Page P-locks are still required when page level locking is used to handle space map pages and index leaf pages (refer to 10.7.2, “Other usage of page P-locks” on page 332 for more information).

Because of the possible increase in P-lock activity with row locking, evaluate row locking carefully before using it in a data sharing environment. If you have an update-intensive application process, the amount of page P-lock activity is likely to increase the data sharing impact. See 12.1, “Database design considerations” on page 418 for more information about lock size row.

It is important to note that page P-locks are only required when the page set is GBP-dependent. If the page set is non-GBP-dependent, page P-locks are not required.

## 10.7.2 Other usage of page P-locks

Page P-locks are also used to maintain consistency for space map pages of GBP-dependent page sets and GBP-dependent index leaf pages.

### Space map page P-locks

Within a table space, space map pages identify the table space's data pages that have enough free space for additional data to be inserted. It also contains information about which pages have been modified since the last image copy. The number of pages that are covered by a space map page depends on the type of table space and the segsize being used. For example, a space map page of a non-UTS partitioned 4K table space covers 10,760 pages.

In a DB2 non-data sharing environment, all space map page updates are serialized with a local exclusive (X) mode page latch. The page latch is released as soon as the physical update to the space map page has completed.

In a DB2 data sharing environment, space map page updates are serialized for all transactions across all sharing members using a page P-lock. To provide consistency, a space map page can only be updated when both the local page latch and the page P-lock on the space map page are held in exclusive mode.

In a DB2 data sharing environment with a high number of INSERT, UPDATE, and DELETE operations, hot spots on space maps can occur because the space management process tries to optimize free space allocation (Figure 10-14).

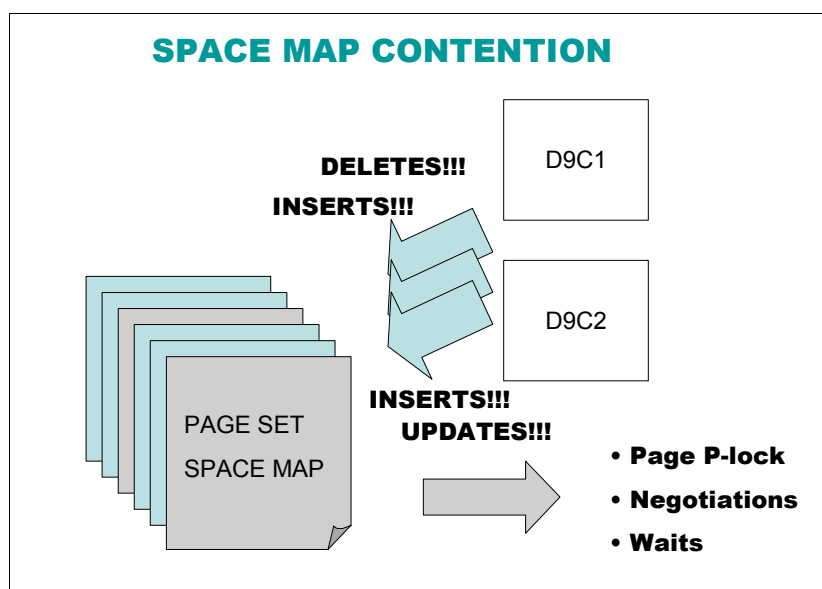


Figure 10-14 Space map page contention

In an environment with a high number of insert, update, and delete operations on multiple members on the same objects, there can be a lot of P-lock negotiations between DB2 members on space map pages. The page P-lock is owned by a DB2 member and can only be held in exclusive mode by one member at a time. For inter-DB2 member serialization, an owning DB2 member can give up the ownership of the page P-lock when the lock is requested by another member.

Refer to “MEMBER CLUSTER” on page 420 for more information about this topic.

### Index leaf page P-locks

Page P-locks are also acquired to serialize index leaf page updates between multiple members in a data sharing environment. This is done under a DB2 latch in a non-data sharing environment.

The index space page set has to be GBP-dependent for page P-locks to be used. The first thread in a member that needs to process the page requests the page P-lock on the index leaf page. The thread is suspended waiting for a successful negotiation of the P-lock for the index leaf page when the page P-lock is already held (owned) by another member at that time.

### 10.7.3 Other types of P-locks

DB2 data sharing does not perform the EDM consistency through a global EDM pool and each DB2 member owns their individual EDM pool. For maintaining the consistency, DB2 uses global P-locks on the EDM pool elements, such as DBDs, sections of plans and packages, and some other control blocks. They are acquired when the EDM element is read into the EDM pool and are released when the item is removed from the pool. When a DB2 member changes an item, it notifies other members through XCF that are also using that DBD that new transactions should use the new DBD, which is read into the EDM pool. P-locks in EDM pool elements have typically low activity and are usually long-lived. A busy DB2 data sharing installation can have thousands of EDM pool active P-locks.

Other types of P-locks are index tree P-locks, castout P-locks, and SKCT/SKPT P-locks.

## 10.8 Pseudo-close and the GBP-dependency

In 10.5, “How inter-DB2 data coherency or read-write interest works” on page 315, we discuss how the process of page set P-lock negotiation works, when inter-DB2 member interest starts, and how objects become GBP-dependent. Using the page set P-locking negotiation mechanism, a DB2 member can dynamically track the level of interest for the page set when physical data access characteristics change depending on the workload.

In this section, we discuss the pseudo-close process and how objects get out of GBP-dependency.

The term *pseudo-close* refers to the point in time when the DB2 determines that the page set has not been updated recently and converts the DB2 member's physical interest on the page set from the R/W state back to the R/O state. By converting the member's interest back to an R/O state, DB2 can narrow the range of log records that must be scanned for media recovery or restart recovery for the page set. The pseudo-close process is controlled by the PCLOSET and PCLOSEN installation parameters.

In a DB2 data sharing environment, the pseudo-close process is also used to dynamically track the inter-DB2 interest levels and getting out of GBP-dependency for a page set. When a GBP-dependent page set has not been updated for a while, DB2 reduces the interest level, to reduce the data sharing impact associated with GBP coherency maintenance and child lock propagation.

To illustrate the process, we use a couple of scenarios:

- ▶ A single member loses interest in a page set
- ▶ Multiple members losing interest



## 10.8.1 A single member loses interest in a page set

Figure 10-15 continues the scenario from Figure 10-8 on page 321 where both members have write interest in a page set, or have inter-DB2 write/write interest.

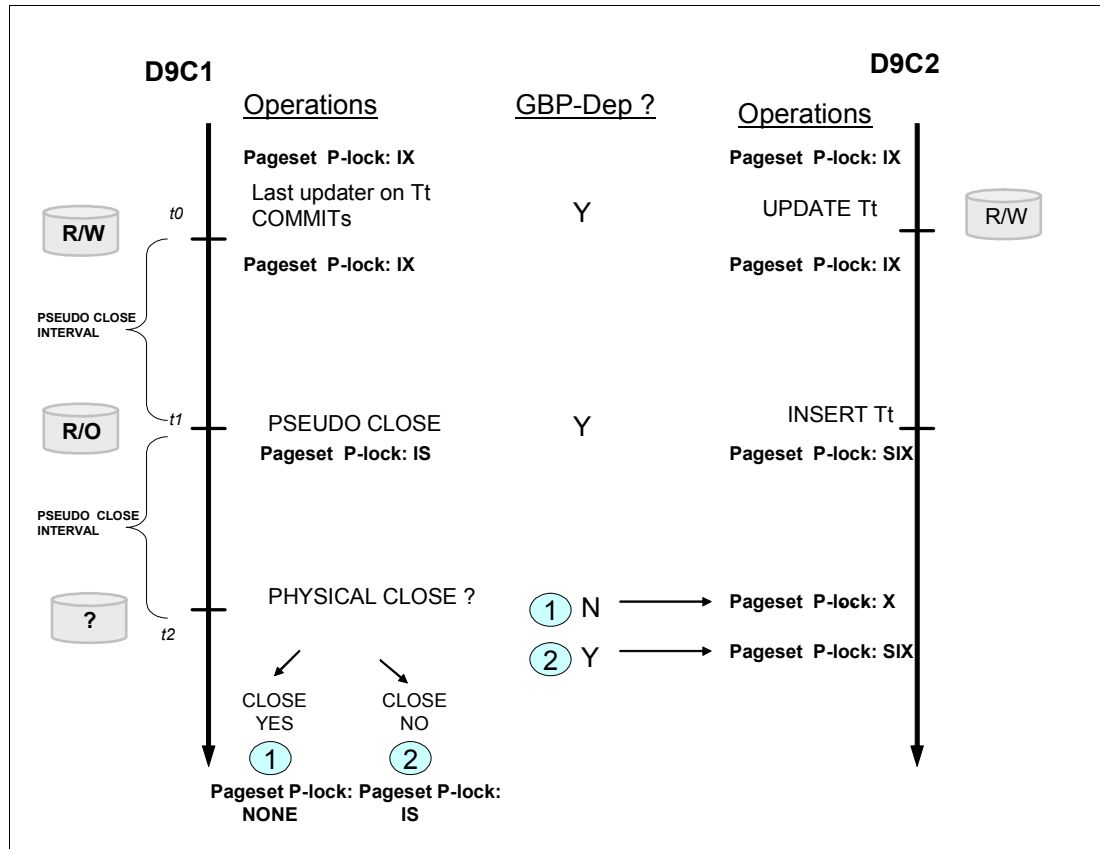


Figure 10-15 Pseudo-close processing for a single member going through pseudo-close

Both DB2 members D9C1 and D9C2 hold an IX page set P-lock after page set P-Lock negotiation.

Note the following items in Figure 10-15:

- ▶ At time t0, the last updating transaction on member D9C1 performs a commit against table Tt.
- ▶ After a pseudo-close interval, at time t1, and no transaction has updated the page set on this member since, the page set is switched from R/W to R/O. At the first pseudo-close interval (t1), the page set P-lock is changed from IX to IS page on D9C1 and changes from IX to SIX on D9C2 after negotiation through the P-lock exit.
- ▶ After a second pseudo-close interval at t2, and provided that the page set has not been accessed since, the page set is either already physical closed and gets out of GBP-dependency, or it has been maintained in R/O mode and stays GBP-dependent, depending on whether the table space or index is defined as CLOSE YES or NO. At time t2, the page set P-lock will maintain its IS state on D9C1 and SIX on D9C2 if the object is defined as CLOSE YES, or go out of GBP-dependency by physically closing the page set and releasing the page set P-lock on D9C1, and change the page set P-lock on D9C2 to X mode.

The physical close process does not necessarily require a second pseudo-close interval. It can happen at any time after the pseudo-close.

**Important:** It is important to understand that even though all data is shared across the members of a data sharing group, each DB2 member performs its own open, pseudo-open, pseudo-close, and close processing for each page set it accesses.

## Implications of using CLOSE NO

Note that since APAR PQ69741, DB2 data sharing honors the CLOSE YES or NO attribute of the table space or index.

For both CLOSE YES and CLOSE NO page sets, DB2 automatically converts infrequently updated page sets or partitions from a read-write to a read-only state (pseudo-close) according to the values you specify in the PCLOSEN and PCLOSET DSNZPARMs.

However, CLOSE NO page sets remain open until closed by a DB2 shutdown, the -STOP DB command, or by reaching the DSMAX, where all candidate CLOSE YES data sets have already been closed. This also means that the page set P-lock will be kept (in IS or S mode, depending on the other system's interest).

On the other hand, using CLOSE NO does not mean that the page set remains GBP-dependent forever, as the page set P-lock can change to S mode, provided the other members stop updating the page set for at least one pseudo-close interval, but it does mean that the moment another member starts updating the page set again, the page set will become GBP-dependent again.

If the object had been defined as CLOSE NO, the page set would have been physically closed after a period of inactivity, and the page set P-lock would have been released. If the other member starts updating again, that member will acquire an X mode page set P-lock, but the object will not become GBP-dependent at that point. Only when a second member starts reading or updating again will the object become GBP-dependent.

## 10.8.2 Multiple members losing interest

Figure 10-16 shows what happens to each DB2 member when both members go through a number of pseudo-close intervals, the P-lock negotiations between them, and their page set P-lock state.

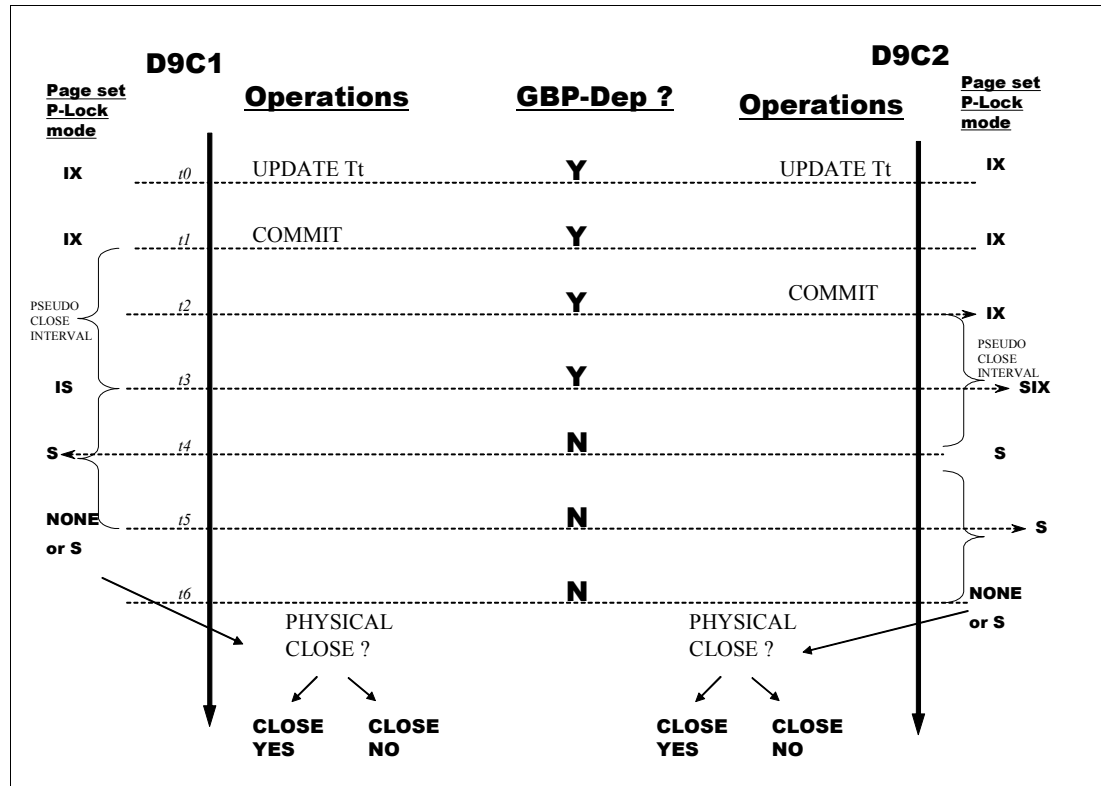


Figure 10-16 Multiple members losing interest

Note the following items in Figure 10-16:

- ▶ At time t0, both DB2 members D9C1 and D9C2 are performing table Tt data updates, and both members hold an IX page set P-lock on the page set that contains table Tt.
- ▶ At time t1, D9C1's last transaction is accessing Tt commits. At this point, the D9C1's page set P-lock mode does not change and remains IX.
- ▶ At time t2, D9C2's last transaction is accessing Tt commits. Again, at this time the page set P-lock mode does not change and remains IX.
- ▶ At time t3, after a pseudo-close interval (PCLOSET/N) has passed on DB2 member D9C1, without any update activity against the page set that contains table Tt, the page set is switched from R/W to R/O.

At this point, the pageset P-lock mode changes from IX to IS on D9C1 and the P-lock exit is driven on D9C2, which triggers the pageset P-lock to change from IX to SIX (the pseudo-close interval has not expired on D9C2 yet)

At this time, the page set still is GBP-dependent.

- ▶ At time  $t_4$ , after the first pseudo-interval on DB2 member D9C2 expires, D9C2 downgraded its page set P-lock from SIX to S (going from read/write to read only, and being the last updater to stop updating the page set), and drives the P-lock exit on D9C2, which triggers D9C1 to downgrade from IS to S (as the last updater just went away).  
At this point, the page set gets out of GBP-dependency. There is no more inter-DB2 read-write interest.
- ▶ At time  $t_5$ , after another PCLOSET/N interval with no activity on the page set, depending on the table space CLOSE YES/NO attribute, it will be physically closed (and release the page set P-lock lock) (— CLOSE YES), or the page set will remain OPEN, and keep the page set P-lock in S mode or state (— CLOSE NO).
- ▶ At time  $t_6$ , member D9C2 goes through the same process as D9C1 at  $t_5$  and also changes the page set P-lock mode according to the CLOSE attribute.

### 10.8.3 Reading member quits first

Figure 10-17 shows a scenario where the interest from one member goes away while the page set is still GBP-dependent. This scenario is somewhat more complicated than the previous one.

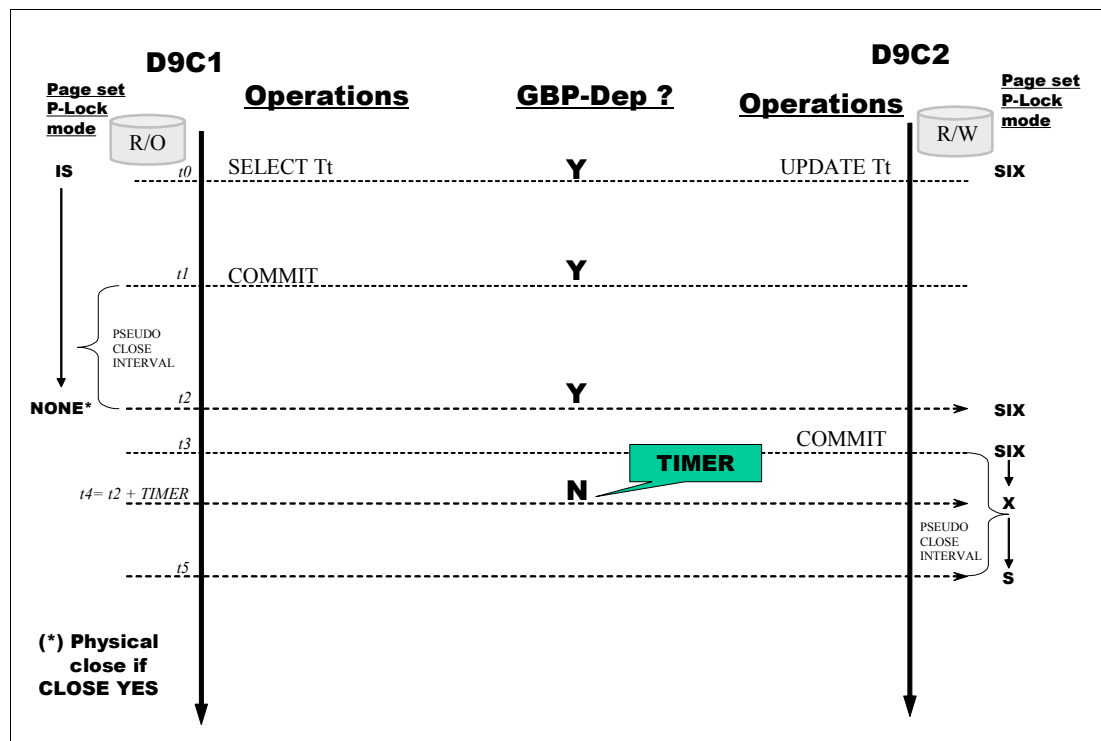


Figure 10-17 Reader quits first

At the start of the scenario, at time  $t_0$ , member D9C1 holds an IS page set P-lock on the page set that holds table  $T_t$ , as it only has a single transaction running against it that is selecting from the table. Member D9C2 holds the page set P-lock in an SIX state, as it has a single transaction that is updating table  $T_t$ .

At time  $t_1$ , the reading transaction on D9C1 commits, and the pseudo-close interval starts.

At time t2, after a pseudo-close interval (PCLOSEN/T) has expired, and since the table space was in a read only state already and has not been accessed at all during the pseudo-close interval, and the object is defined with CLOSE YES, the page set is physically closed and the P-lock is released. At this point, the P-lock exit is driven on D9C2 and changes the cached state of the P-lock to X, but the held state remains unchanged at SIX, and the page set remains GBP-dependent at this point.

At time t3, the updating transaction on D9C2 commits and releases its L-locks and claims. At this time, the object still remains GBP-dependent, as a commit does not change the state of the page set P-lock.

DB2 has a service task that wakes up every 120 seconds and check the cached versus the held state of the page set P-locks. At time t4, the page set that holds table Tt is still GBP-dependent (and member D9C2 would continue to propagate its child locks to XES if a new transaction started accessing table Tt).

When the timer driven service task starts its processing, it notices that the cached state for the table that holds the table is X, while the held state is SIX. As there are currently no claimers on the page set, it can be successfully drained and the page set P-lock on D9C2 is downgraded to X. At this point, the page set drops out of GBP-dependency and will also stop propagated child L-locks.

At time t5, after a pseudo-close interval has expired on D9C2, the inter-DB2 read/write interest can be downgraded from RW to RO, and the page set P-lock is downgraded to S.

Note that only one pseudo-close interval has passed when D9C1's last reader transaction commits the work in time t2. As the transaction in member D9C1 is a reader, the page set is already in R/O mode.

## 10.8.4 Pseudo-close considerations

The pseudo-close process helps to reduce the interest level that page sets on different members have in the DB2 object they use. This is good news, as reducing the interest level also reduces the data sharing impact.

On the other hand, getting into GBP-dependency is an expensive process, as pages need to be registered, updated pages have to be written to the GBP, and child locks need to be propagated. Therefore, we recommend that a page set does not go into and out of GBP-dependency all the time. We do not recommend that a page set drops out of GBP-dependency when it will become GBP-dependent again one minute later. Also, in many cases, it is not just a single object that becomes GBP-dependent again. This usually happens when activity for a certain application picks up again, and a large number of page sets become GBP-dependent almost at the same time. This can cause major spikes in the processor usage, for both the z/OS system as well as the CF, when all of these requests to become GBP-dependent have to be processed simultaneously.

So you must strike a good balance between reducing the data sharing processing impact by allowing a page set to drop out of GBP-dependency, and the processing impact it takes to go in and out of GBP-dependency.

To control GBP-dependency you can use:

► DSNZPARMs that control pseudo-close

– System checkpoint (CHKFREQ DSNZPARM)

Determines the number of minutes, or the number of log records before DB2 takes a system checkpoint. The value will help determine how often a pseudo-close based on PCLOSEN is driven. In addition, the pseudo-close processor is driven at system checkpoint (to check whether some objects qualify for read-only switching based on PCLOSEN and PCLOSET).

– RO SWITCH TIME (PCLOSET DSNZPARM)

Indicates the number of minutes since a page set or partition was last updated, after which DB2 will convert the page set or partition from read-write to read-only.

– RO SWITCH CHKPTS (PCLOSEN DSNZPARM)

Indicates the number of consecutive DB2 system checkpoints since a page set or partition was last updated, after which DB2 will convert the page set or partition from read-write to read-only.

► CLOSE parameter on the CREATE TABLESPACE and CREATE INDEX

Since APAR PQ69741, DB2 honors the CLOSE attribute on the CREATE TABLESPACE and CREATE INDEX, which means that CLOSE NO objects are not physically closed during pseudo-close processing. This can mean that the object remains GBP-dependent for a longer period of time, or that an object can become GBP-dependent again earlier than would be the case for CLOSE YES objects. For more information, refer to “Implications of using CLOSE NO” on page 336. The recommendation is to use CLOSE NO for objects that are GBP-dependent most of the time. For page sets that are GBP-dependent infrequently, CLOSE YES would be the better choice.

► -ACCESS DB MODE(NGBPDP)

As there is a processing impact associated with DB2 data sharing in terms of GBP activity and additional locking activity, we recommend making an object non-GBP-dependent when the object is only going to be accessed from a single member for a certain period of time. A typical example is during the batch window. During the batch window, a number of jobs have to be run against a number of objects, but the work against a set of objects can run on a single member. Batch jobs also tend to process large quantities of data that require large number of getpages, updated pages, and locks. In such a situation, it is beneficial to make sure that the objects that are used by these jobs are not GBP-dependent, so no locks need to be propagated, and no pages have to be sent to the GBP.

DB2 9 for z/OS introduced a new command to remove the GBP-dependency for a page set:

```
-ACCESS DATABASE () SPACENAM () MODE(NGBPDEP)
```

An important consideration in DB2 data sharing is to tune the PCLOSET and PCLOSEN DSNZPARMs to reduce the processing impact of getting in and out of GBP-dependency very frequently. When an object becomes GBP-dependent, there is extra work in the cache control mechanism and the page set P-locks negotiation. This includes registering pages, caching the pages, and propagating the child locks. This is synchronous to the transaction that triggers the object to become GBP-dependent, in other words, the transaction will wait for this process to complete.

When a member removes the GBP-dependency for an object, the updating member must externalize the changed pages, adding the cost of cast out the pages. As the pseudo-close process runs as a system task, the CPU time it uses is charged to the DBM1 address space.

**Tip:** When analyzing a DB2 statistics report, try to keep “DSETS CONVERTED R/W -> R/O” to 10 to 15 per minute. (Unfortunately, there is no indication about how many objects that went through read-only switching were GBP-dependent.)

## 10.9 Managing the DB2 lock structure

In this section, we discuss how to size, define, allocate, and manage the DB2 lock structure. We also explain how retained locks are handled.

### 10.9.1 Definition of the lock structure

You use the z/OS coupling facility resource management (CFRM) policies to define the lock structure (like any for other type of structure in a Parallel Sysplex). A CFRM policy determines how and where the structure resources are allocated. There is only one active coupling facility resource management (CFRM) policy per Parallel Sysplex.

Example 10-4 shows a sample definition for the lock structure. We discuss how to size the lock structure in next section. The purpose of this section is to highlight some of the lock structure options.

#### *Example 10-4 Lock structure definition*

---

```
STRUCTURE NAME=DB9CG_LOCK1
INITSIZE=8192
SIZE=16384
ALLOWAUTOALT(YES)
FULLTHRESHOLD=80
MINSIZE=8192
PREFLIST=(CF1,CF2)
REBUILDPERCENT(1)
```

---

We only describe some of the lock structure options in this section (for a more formal definition of all the parameters, refer to *z/OS V1R10.0 MVS Setting Up a Sysplex*, SA22-7625):

- ▶ **INITSIZE:** Refer to the (also for **SIZE**) sizing recommendations for the lock structure in “Lock table size” on page 342.
- ▶ **SIZE:** This is the max size allowed for the structure. See **ALLOWAUTOALT**.
- ▶ **ALLOWAUTOALT:** This option specifies implementing the Auto Alter z/OS capability<sup>2</sup>. Auto Alter has algorithms that can request that the CF dynamically increase or decrease the number of entries or data page elements to avoid structure full conditions. For the lock structure, it can only dynamically increase the RLE portion of the structure.

---

<sup>2</sup> Auto Alter capability is initiated by the z/OS structure full monitoring that occurs regularly for each structure. This monitoring determines the level of usage for objects within a coupling facility, and issues a warning message if a *structure full* condition is imminent. Doing this allows tuning actions to avoid a structure full condition. Structure full monitoring occurs every few seconds. Auto Alter has algorithms that can request the coupling facility to dynamically increase or decrease the number of entries or data page elements to avoid structure full conditions. It can increase or decrease the size of the structure if necessary.

- **MINSIZE:** This option allows Auto Alter to decrease the structure size when the CF is under storage stress, that is, less than 10% free CF storage for structures. You should use at least INITSIZE in this option.
- **PREFLIST:** This option tells the CFs where a structure should be placed and the preferred location for the structure. If a CF failure occurs, either due to failure of the CF or the failure or CF links, the secondary structure (CF2) is chosen for structure.
- **REBUILDPERCENT:** This option describes the percentage of loss of connectivity that is tolerated before XES rebuilds the structure. Use 1%, because you would always want the rebuild to occur on the other CF if any loss of connectivity is detected.

**Important:** There should be white space in the CF1 and CF2 coupling facility to allocate all of the structures. In the rare event that a coupling facility fails (for whatever reason), the structures can rebuild quickly in the remaining coupling facility. This is specified in the STRUCTURE statement by way of the REBUILDPERCENT value of 1, and the PREF keyword, where at least one other coupling facility is referenced.

## 10.9.2 Lock table size

When IRLM allocates the lock structure, it requests the structure size (INITSIZE) from XES and attempts to divide the lock structure storage evenly between the lock table portion and the record list portion. This even split is done if the NUMBER OF LOCK ENTRIES(LTE) field in the DSNTIPJ installation panel is set to 0.

The size of the lock table always has to be a power of two (with respect to the number of bytes); the even 1:1 storage distribution between lock table and record list is not always possible if the lock structure size is not a power of two. If a 1:1 split occurs and the total size is not a power of 2, you might experience a severe shortage of space for the record table entries, resulting in DB2 or possible IRLM failures.

To avoid confusion and possible surprises in the actual lock table or record list sizes, it is best to specify the size (INITSIZE) of the lock structure as a power of two. Table 10-8 shows the recommended values for the lock structure size. Our recommendation for a 2-way data sharing is to start with a safe INITSIZE value of 64 MB. See *DB2 for z/OS: Data Sharing in a Nutshell*, SG24-7322 for a discussion about these values and the use of the CFSizer.

Table 10-8 Recommendations for the lock structure

INITSIZE	SIZE	Condition
16 MB	32 MB	For light sharing with a low amount of updating (or single member data sharing)
32 MB	64 MB	For medium sharing with a moderate amount of update activity
64 MB	128 MB	For high amount of sharing with a high amount of update activity

You can control the division of the lock structure storage between these two components by using the IRLMPROC number of lock entries parameter (LTEs). The LTE specification is done during the DB2 subsystem installation or by using the MODIFY irlmproc,SET,LTE command. When you configure the LTE value, you determine the size of the lock table in the coupling facility lock structure. The remainder of the size becomes the RLE<sup>3</sup> or Modify lock list part of the structure.

<sup>3</sup> The number of record table entries that were available in the coupling facility the last time this IRLM was connected to the group.



The LTE value must be a power of 2 in the range of 1 to 1024, with each increment representing 1,048,576 lock table entries. So, the range of the values are [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, and 1024]. IRLM limits the number of lock table entries to a maximum of 1,024 MB.

For example, if LTE is specified to 128, there will be 134,217,728 (128 multiplied by 1,048,576) lock table entries. Refer to *DB2 Version 9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845 for more information about the storage estimate for the lock structure.

The DISPLAY GROUP command in Example 10-5 shows the size of the lock structure, the number of lock table entries, and the list entries.

*Example 10-5 DISPLAY GROUP command output*

---

```

DSN7100I  -D9C1 DSN7GCMD
*** BEGIN DISPLAY OF GROUP(DB9CG  ) GROUP LEVEL(910) MODE(N )
                PROTOCOL LEVEL(2)  GROUP ATTACH NAME(D9CG)
-----
DB2                DB2 SYSTEM      IRLM
MEMBER  ID  SUBSYS  CMDPREF  STATUS  LVL NAME      SUBSYS  IRLMPROC
-----
D9C1      1  D9C1   -D9C1   ACTIVE  910 SC63      I9C1   D9C1IRLM
D9C2      2  D9C2   -D9C2   ACTIVE  910 SC64      I9C2   D9C2IRLM
-----
SCA  STRUCTURE SIZE:      8192 KB, STATUS= AC,   SCA IN USE:      4 %
LOCK1 STRUCTURE SIZE:      8192 KB
NUMBER LOCK ENTRIES:      2097152
NUMBER LIST ENTRIES:      11111, LIST ENTRIES IN USE:      8
*** END DISPLAY OF GROUP(DB9CG  )
DSN9022I  -D9C1 DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION

```

---

In the example, the coupling facility lock structure is 8 MB, and as we have NUMBER OF LOCK ENTRIES(LTE) equal to 0, both the lock table and record list table will 4 MB each. The LOCK ENTRY SIZE was set to 2 (MAXUSRS equal to 7) during the installation time, so we will have 2 MB of number of lock entries (4 MB divided by 2). Refer to Chapter 11, “Monitoring data sharing locking activity” on page 365 for more information.

## Altering the size of lock table

Because structure allocation is done at CONNECT, any change that is made to the LTE= parameter value does not take effect unless the group is terminated, structure forced, and the group restarted, or a REBUILD is done.

When a global lock request is made, IRLM assigns locked resources to an entry value in the lock table. IRLM hashes different db2 resource to the same lock entry, so there is no contention. Thus, false contention can also occur if the lock table is small. The lock structure may be small because you defined the lock structure to be small or because when initially defined the lock structure in the coupling facility and defined the lock table entries (LTE) parameter in the IRLMPROC as 0, the underestimated one half of the lock structure space was assumed.

If false contention has been identified by two lock requests hashing to the same locktable entry, the global lock manager XES must resolve the contention. The other XESs in the group are instructed to send the lock information they have that pertains to the lock table entry to the global lock manager XES so it can determine whether the contention is real or false. Refer to 10.10, “Global contention” on page 353 for more information.

The locking protocol 2, introduced in DB2 V8 NFM, has reduced tremendously the level of false contention. The reason is that IX-L locks are mapped to S XES locks, which are compatible locks. The lock requests do not need to be suspended, as fewer lock table entries are occupied by X-XES locks and false contention is thereby reduced. Refer to *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465 for information about the V8 protocol level 2.

The monitoring of the false contention can be performed by using DB2 statistics and accounting traces and the RMF report.

The lock table size cannot be increased dynamically. To increase the lock table size, you can either:

- ▶ Increase the LTE value
- ▶ Increase the lock structure within the CFRM Policy

### ***Increase the LTE value***

Specifying the LTE= gives you control over the lock table size. This control is particularly beneficial if, over time, your lock structure grows, but your lock list does not increase. Continually doubling the lock structure size only for the purpose of increasing the LTEs wastes storage. Assuming that the CFRM policy INITSIZE (or you still can grow to SIZE) of the lock structure is enough and you want to increase the number of lock entries, perform these steps:

1. Use the z/OS command to dynamically increase to SIZE:

```
SETXCF START,ALTER,STRNAME=DB9CG_LOCK1,SIZE=newsize
```

2. Modify the LTE parameter in the IRLMproc using this command:

```
F ir1mproc,SET,LTE=x
```

3. Rebuild the lock structure using this command:

```
SETXCF START,REBUILD,STRNM=strname,LOC=NORMAL|OTHER
```

Each LTE=x value represents a megabyte number of entries. Refer to Chapter 53. “MODIFY ir1mproc,SET (z/OS IRLM)”, in *DB2 Version 9.1 for z/OS Command Reference*, SC18-9844, for detailed information about setting the LTE lock table entries values. You can dynamically increase the structure sizes up to a maximum limit SIZE specified on the CFRM policy by using the z/OS command SETXCF START,ALTER, if If CFLEVEL is greater than 0.

### ***Increase the lock structure within the CFRM policy***

If the SIZE in the CFRM policy is still not big enough, you must increase the lock storage in the CFRM policy and rebuild the lock structure. Perform the following steps:

1. Increase the storage for the lock structure in the CFRM policy INITSIZE and SIZE parameter. Remember that it must be a power of two.
2. Run IXCMIAPU to create the new policy.
3. Use the z/OS command SETXCF START,POLICY to start the updated policy.
4. Use the following z/OS command to rebuild the structure:

```
SETXCF START,REBUILD,STRNAME=DSNDBOA_SCA
```

### 10.9.3 Lock table entries

The LOCK ENTRY SIZE parameter in the DSNTIPJ installation panel determines the width of a lock entry in the lock table and the size can be 2, 4, or 8 bytes. DB2 converts the value for LOCK ENTRY SIZE to a corresponding value for the IRLMPROC parameter MAXUSRS, as shown in Table 10-9.

*Table 10-9 Lock entry size and corresponding MAXUSRS*

Lock entry size	MAXUSRS values
2	1-7
4	8-23
8	24-32

If you have less or equal than six DB2 members in your DB2 data sharing group, setting MAXUSRS parameter to a higher value than the required, for example, from MAXUSRS > 8, will make the lock table use the 4 bytes lock table entries. Half of the space allocated for the lock table will not be used. Only two bytes are used, one for the global byte and one byte for the usage of share bits, so be sure to specify the proper lock table width.

When the lock structure was allocated based on a MAXUSRS value of 7 (width of 2 bytes), IRLM will automatically rebuild the lock structure to a 4 byte width (23 user share bits) when the seventh member joins the group. A lock structure rebuild is not necessary when the seventh member joins the group. Even if you anticipate your group growing beyond seven members, you can start out with a lock entry size of 2 and make the most efficient use of lock structure storage.

The recommendation is that when running a data sharing group of 6 members or less, the MAXUSRS value should be 7 or less to get the minimum lock table entry width of two bytes. This will maximize the number of lock table entries for a given coupling facility lock structure size.

You can change the MAXUSRS in the IRLMPROC procedure for the IRLM instance that is associated with each member of the data sharing group. Lock structure allocation occurs when the first member joins the data sharing group after an IPL of the operating system, or after any situation that causes the lock structure to be deallocated. The values specified in its MAXUSRS will be used for determining the width of lock table entries. Note that rebuilding the structure will not alter the lock entry size.

For example, if you have six members in your DB2 data sharing environment and MAXUSRS ≤ 7, and you are planning to activate two more members, you should, in advance, increase the lock structure in size and change the MAXUSRS of all your DB2 members from 8 to 23. If an IPL is made or the lock structure is deallocated before adding the two new members, when the first DB2 member (which MAXUSRS between 8 and 32) joins the data sharing group, it will cause the table lock table entry to have a width of 4 bytes. If these alterations are not made in advance, then when the seventh member is added into the data sharing group, DB2 IRLM will automatically rebuild the lock structure and have a 4 byte entry width. IRLM prepares the lock structure to handle an eighth member joining the group. Be aware that the rebuild should occur in an off-peak period.

If you increase the lock entry size (and thereby increase MAXUSRS or by the adding of a new member), increase the lock structure size to maintain the number of lock table entries and record list entries. If you do not increase the lock structure size, IRLM obtains the storage that it needs for the increased lock entry size from storage for record list entries.

If you change the IRLM startup procedure directly, the parameter you change is called MAXUSRS. The value of LOCK ENTRY SIZE is translated during the DB2 installation or migration process.

The next example is to help you understand why we change the lock entry width. We change the MAXUSRs of our two DB2 members to MAXUSRs=23. The INITSIZE and SIZE of lock structure is maintained. A START DB2 command is issued for DB2 member D9C1, and when the IRLM I9C1 connects, we can see the IXC582I message in SYSOUT log, as shown in Example 10-6.

*Example 10-6 IXC582I message in SYSOUT log*

---

```

IXC582I STRUCTURE DB9CG_LOCK1 ALLOCATED BY SIZE/RATIOS. 635
  PHYSICAL STRUCTURE VERSION: C4952AOA ABF1C409
  STRUCTURE TYPE:                LOCK
  CFNAME:                        CF1
  ALLOCATION SIZE:                 8192 K
  POLICY SIZE:                   16384 K
  POLICY INITSIZE:               8192 K
  POLICY MINSIZE:                 0 K
  IXLCONN STRSIZE:               0 K
  ENTRY COUNT:                   10445
  LOCKS:                         1048576
  ALLOCATION SIZE IS WITHIN CFRM POLICY DEFINITIONS
  IXL014I IXLCONN REQUEST FOR STRUCTURE DB9CG_LOCK1 636
  WAS SUCCESSFUL.  JOBNAME: D9C1IRLM ASID: 0093
  ....
DXR132I I9C1001 SUCCESSFULLY JOINED THE DATA SHARING GROUP WITH      1M
LOCK TABLE ENTRIES AND      10445 RECORD LIST ENTRIES

```

---

The size of our lock table is 8 MB, and as LTE=0 in IRLMPROC, DB2 evenly divides both the lock list table and record list table into 4 MB each. As our lock entry width is 4 bytes, we get space for 1,048,576 (1 MB) number of lock entries. The DISPLAY GROUP command output is shown in Example 10-7.

*Example 10-7 DISPLAY GROUP command output*

---

```

DSN7100I  -D9C1 DSN7GCMD 694
  ....
SCA  STRUCTURE SIZE:      8192 KB, STATUS= AC,   SCA IN USE:      4 %
LOCK1 STRUCTURE SIZE:     8192 KB
NUMBER LOCK ENTRIES:    1048576
NUMBER LIST ENTRIES:      10445, LIST ENTRIES IN USE:           0
  ....
DSN9022I  -D9C1 DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION

```

---

### ***Decrease of lock table entry width***

IRLM does not automatically rebuild if the number of members decreases. To decrease the lock entry size, you have always to perform a disruptive work, as described in *DB2 Version 9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845.

Perform these steps:

1. Quiesce all members of the group using the following command:  
     -STOP DB2 MODE(QUIESCE)

2. If IRLM is not automatically stopped along with DB2, enter the z/OS command:  
`-STOP irلمproc`
3. Optionally, use the command `D XCF,STRUCTURE` to display the structure status, and force the deallocation of the lock structure by issuing the following z/OS command:  
`SETXCF FORCE,STRUCTURE,STRNAME=strname`  
 This will also remove any failed persistent connections to the structure, if they exist.
4. Change the lock entry size `MAXUSRS` for at least one IRLM. (You should change the value for all of them.)
5. Start the member and IRLM that have the updated value. (You must start the updated member first.)
6. Start all other members.

A group restart occurs when you restart the members. Because you quiesced work before changing the lock entry size, the group restart should be relatively quick.

IRLM assigns locked resources to an entry value in the lock table. This is how it can quickly check to see if a resource is already locked. If the lock structure is too small (thereby making the lock table portion too small), many locks can be represented by a single value and false contention occurs.

The total size of the lock table in the coupling facility lock structure must be large enough to limit hash contention, which prevents performance problems. Proper specification for the number of lock table entries can help avoid false contention. Refer to 10.10.3, “False contention” on page 354 for more information.

## 10.9.4 Record list table

You can increase the size of a record list table dynamically if there is still enough space by using the following command:

```
SETXCF START,ALTER,STRNAME=DB9CG_LOCK1,SIZE=newsize
```

You do not need to rebuild the coupling facility lock structure if the `SIZE` parameter solves your problem of lack of space on the record list table. Auto Alter has algorithms that can request that the CF to dynamically increase or decrease the number of entries or data page elements to avoid structure full conditions.

If there is no lock structure size or enough room to increase, you have to increase the lock structure within the CFRM policy, as described in “Increase the lock structure within the CFRM policy” on page 344.

The lock structure must also be large enough to prevent failures that result from a lack of record table storage to write a modify entry in the record list table. For the record list table, IRLM reserves 10% of the record table entries for “must complete” functions (such as rollback or commit processing), so that a shortage of storage does not cause a member to fail.

If storage runs short in the record table, there can be an impact on availability (transactions are terminated), response time, and throughput. When the storage fills, an application starts to receive an “unavailable resource” message with reason code 00C900BF for L-locks and 00C20257 for P-locks.

The DISPLAY GROUP in Example 10-5 on page 343 shows the total number of list entries and the number of list entries that are in use. You should regularly monitor these values, as (if the record list table fills up) your DB2 installation will not be able to process data modification requests.

### Record list storage shortage

When the record list begins to fill, a DXR170I message indicates when storage reaches 50, 60, and 70% full. The DXR142E message appears as the thresholds are reached, starting at 80% full:

```
-D9C1 DXR142E THE LOCK STRUCTURE structure-name IS zzz% IN USE
```

This message will stay on the console until the operator removes the message, or until storage falls below 70% in use.

Lock structure record list storage shortages are a serious problem and need to be corrected immediately. Update locks requests will fail when the record list is full and the applications will receive “resource unavailable” messages.

You can correct the record list storage failure by using one of these solutions:

- ▶ Restart the DB2 members that are down and holding retained locks.
- ▶ If the lock structure in the coupling facility has enough storage, use the z/OS XCF ALTER command to increase the SIZE. If you are using the auto alter z/OS capability, view the lock structure in the coupling facility to see if you have already achieved the maximum SIZE limit.
- ▶ Lower the lock escalation values to get fewer locks. You can do this task by either lowering the value on the LOCKS PER TABLE(SPACE) of installation panel DSNTIPJ or by using the LOCKMAX clause of CREATE TABLESPACE.
- ▶ Increase the size of the lock structure, as described in “Increase the lock structure within the CFRM policy” on page 344.

When a full condition is rapidly occurs in your installation, the immediate action is to provide IRLM with more lock structure storage, either through the addition of more storage (using the SETXCF START,ALTER command) or through the reduction of uncommitted transactions.

When the storage is free up to 70%, the following message appears:

```
-D9C1 DXR169I THE STORAGE SHORTAGE FOR LOCK STRUCTURE DSNPBRE_LOCK HAS BEEN  
RELIEVED
```

**Note:** Auto Alter is designed for gradual growth, not for spikes in the workload.

## 10.9.5 Modify and retained locks

In the case of a member failure, DB2 update data locked by those transactions that were in progress that had not yet reached a point of consistency at the time of failure must be protected. DB2 data sharing uses modify locks and retained locks to provide this protection.

A modify lock is a lock held on a resource that is in the process of being updated or modified. If a DB2 member fails, then all the modify locks held by the DB2 member at the time of the failure are converted into retained locks.

A modify lock is an active X type (X, IX, or SIX) page set or page P-lock and L-lock.

Retained locks are owned by the DB2 member's IRLM and not by the transactions. These locks are stored here regardless of whether the resource is GBP-dependent.

The record list contains a list header for each DB2 member. Each list is a chain of "list entries" representing modify locks or retained locks owned by that DB2 member. Each list entry is 64 bytes in length. The list entry basically identifies the resource name, the lock state, and whether or not it is a retained lock (refer to Figure 10-5 on page 313).

When DB2 requests a modify lock, IRLM, through MVS XES services, must interact with the coupling facility lock structure for two distinct operations:

- ▶ The lock table must be consulted for inter-DB2 lock compatibility checking.
- ▶ The record list must be updated to track the modify lock on a resource.

These two operations are bundled in one call to the CF and occur synchronously to the requesting task. An interaction with the CF for a modify lock is slightly more expensive than an interaction for a nonmodify lock; a nonmodify lock does not need an entry in the record list.

If the member is currently active, then the record list entries for that member contain all of the modify locks for that member that have been propagated from IRLM to XES. If the member is currently quiesced, then that member will normally have no record list entries.

When a member fails, each surviving IRLM reads the record list to build local copies of the retained locks of the failed member. Each IRLM also keeps a local copy of all the retained locks for fast reference. The redundancy in tracking the retained locks is an important availability consideration. Because of this redundancy, the retained locks can survive a failure of the coupling facility or of all the IRLMs in the group.

Retained locks persist across the failure, and thus can continue to protect database resources that were in an inconsistent state at the time of the failure from being accessed by other DB2 members. Retained locks continue to be held until the failed DB2 member completes its restart recovery and brings the database resources back to a consistent state.

Example 10-8 shows the L-locks and page set /partition P-locks retained locks in failing member D9C2.

*Example 10-8 DISPLAY DATABASE ...LOCKS command: retained*

---

```

-D9C1 DIS DATABASE(RITA1229) SPACE(*) LOCKS
DSNT360I  -D9C1 *****
.....
NAME      TYPE PART  STATUS                      CONNID  CORRID  LOCKINFO
-----
...
GLWSEMP   TS      0001 RW                      R-X,PP
-          MEMBER NAME D9C2
GLWSEMP   TS      0001 RW                      R-IX,P
-          MEMBER NAME D9C2
GLWSPRJ   TS                      RW                      R-IX,PP
-          MEMBER NAME D9C2
GLWSPRJ   TS                      RW                      R-IX,S
-          MEMBER NAME D9C2
GLWXEMP1  IX      0002 RW
GLWXEMP1  IX      0003 RW                      R-X,PP
GLWXEMP1  IX      0003 RW                      R-X,PP
-          MEMBER NAME D9C2
.....
***** DISPLAY OF DATABASE RITA1229 TERMINATED *****

```

---

If another DB2 member attempts to obtain a lock on a resource while there is still an incompatible retained lock on that resource, IRLM immediately rejects the request, and the user receives the “resource unavailable” message.

When a member is running a utility, it holds a lock on the utility ID (UID) for that utility. That lock is also retained if the member fails. This means that you cannot restart a utility until the failed member is restarted and the retained lock is converted to an active lock.

In the process of a restart DB2 member after a failure event, the process of “clearing” the retained lock depends on the type of retained lock:

- ▶ Page set or partition P-lock: DB2 converts the retained locks to active locks. This is called reacquiring the lock. Reacquiring is done when page sets are opened for log apply.
- ▶ L-locks and page P-lock: DB2 purges the lock. L-locks are purged at the end of restart processing page. P-locks are purged at the end of forward recovery.

### **RETLWAIT DSNZPARM parameter**

The RETAINED LOCK TIMEOUT (or RETLWAIT) parameter in DSNZPARM (installation panel DSNTIPI) specifies how long a transaction should wait for a lock on a resource if another DB2 in a data sharing group has failed and is holding an incompatible lock on that resource.

RETLWAIT=0 (the default) indicates that the lock request will be immediately rejected and the “resource unavailable” condition returned to the application. Setting RETLWAIT to a value causes DB2 to suspend the application to wait for the lock to become available. If the request times out, a “timeout” condition is returned to the application. Our recommendation is to set RETLWAIT to an acceptable period in your installation. Refer to *DB2 Version 9.1 for z/OS Installation Guide*, GC18-9846 for more information.



## Retained page set or partition P-lock

A retained P-lock means that other members cannot access the data that the P-lock is protecting if the accessing member requests a P-lock in an incompatible state. For example, if a member fails while holding an IX page set P-lock, it is still possible for another member to obtain an IX page set P-lock on the data.

Retained locks are not needed for resources that were accessed as RO at the time of the failure and these locks can be released for read-only non GBP-dependent page set or partition. For both page set or partition non GBP-dependant with RW interest and GBP-Dependent with inter-DB2 write-write interest in a failing DB2 member, retained lock will be X or NSU, as shown in Table 10-10. Thus, the X P-lock modes should be released as quickly as possible.

Table 10-10 Retained P-lock modes

Interest of failing DB2	Interest of other members	Retained P-lock mode of single member	GBP-dependent
Read-Only	NONE or Read-Only	none	NO
Read-Only	Read-Write	none	YES
Read-Write	NONE	X or NSU	NO
Read-Write	Read-Only	IX <sup>a</sup>	YES
Read-Write	Read-Write	X	YES

a. The P-lock is retained in SIX mode if the page set or partition is an index that is not on a DB2 catalog or directory table space.

You should avoid limiting access to a DB2 resource to a single member, as retained X pageset or partition P-locks prevent access from other members until the failing member is able to restart.

Figure 10-18 shows that D9C2 is a failing DB2 member and that it retained the held P-locks. Member D9C1 has an intent to share P-lock (IS) on page set or partition Tbs A. Member D9C2 has a share with intent exclusive (SIX) on Tbs A and an exclusive (X) P-lock on Tbs B.

When member D9C2 fails, the retained P-lock for Tbs A is intent exclusive (IX) and the page set or partition A can still be accessed by other processes running on member D9C1. The exclusive mode (X) of page set or partition P-lock of Tbs B is not compatible with other P-lock modes, and the access to Tbs B is only allowed when the retained X P-locks are released.

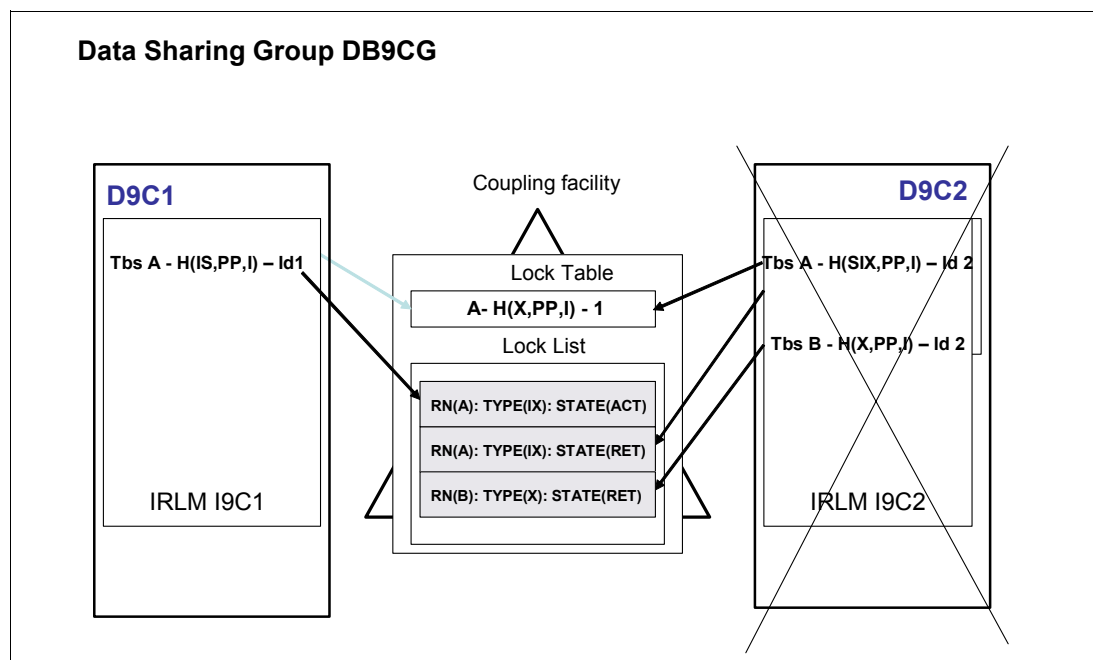


Figure 10-18 Retained page set or partition P-locks after a member failure

## Restart after DB2 failure

Retained locks should be released as soon as possible to allow access to all data by the surviving members of the group.

### RESTART(LIGHT)

If the failed DB2 subsystem was running on a z/OS image that is no longer available, it is critical to restart the failed DB2 in another z/OS image in the same Parallel Sysplex (where another member might be active) in order to release the retained locks. Another z/OS image may not have the resources to handle the workload of an additional DB2 subsystem. RESTART(LIGHT) enables DB2 to restart with a minimal storage footprint to quickly release retained locks and then terminate normally.

### Normal restart

For this type of restart, you simply restart DB2 using automation, either your own or the automatic restart manager (ARM). Automation is much faster than manual operation. The time to restart becomes a critical part of data sharing, because the DB2 members that are active can experience timeouts for a workload due to retained locks held on a failed member. Normal restart for a member of data sharing group is exactly the same as for a single subsystem. For more information about normal restart, refer to “Normal restart for a data sharing member” in Chapter 5, “Operating with data sharing”, in *DB2 Version 9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845.

### **Group restart**

A group restart is initiated by DB2 if either the SCA or the lock structure (or both) are lost (CF failure or deactivation), and cannot be rebuilt on the other coupling facility. When this happens, all members of the data sharing group terminate abnormally. Group restart rebuilds the information from each member's logs. Any member that is restarted will detect the situation and will rebuild the SCA and lock structures, but it is best to start all members at once. Before performing a group restart, you need to delete a persistent structure in the CF or a failed-persistent connection.

## **10.10 Global contention**

In a DB2 data sharing environment, contention can occur when different members need to use the same resource at the same time in a state that is not compatible. If the holder is on a different DB2 member than the requester, this is called *global lock contention*.

If an application is well-tuned to have a low lock contention rate (<2%) in a non-data sharing environment, then when you go to data sharing with this application, the large majority of (global) lock requests should be granted (synchronously) without experiencing any type of global contention.

Global lock contention in a data sharing environment comes mainly in three types. They are listed in increasing order of the time that is needed to resolve them:

- ▶ *False contention* occurs when the hashing algorithm provides the same hash value for two different resources.
- ▶ *XES contention* arises because XES is only aware of two lock modes, share and exclusive (S and X). IRLM locking supports many additional lock modes (S, X, U, IX and so on; there are 11 lock states in total). Therefore, it can appear to XES that contention exists, but when IRLM looks at the lock states, they can be compatible.
- ▶ *Real contention* occurs when lock requests are incompatible, for example, one program holds a logical S lock on a page while another program requests an X lock on the same page. When real contention occurs, the requester remains suspended until the incompatible lock is released or the program abends because of timeout or deadlock.

XES uses the coupling facility lock table to detect possible inter-DB2 lock contention. Contention occurs when different members need to use the same resource (hash entry) at the same time. We explain each type of contention in more detail in the coming sections.

### **10.10.1 Global lock manager**

The process of managing and handling all lock requests that hash to one lock table entry is called *global lock management*. The member that is responsible for managing this hash entry becomes the global lock manager (GLM) for the lock table entry.

When there is contention on a lock with X mode interest in a lock table entry, the XES component of the member holding the exclusive lock becomes the global lock manager for that entry in the lock table. The global lock manager is sometimes also called the owner of the lock table entry or the owner of the hash class. The identification of the global lock manager is inserted into the lock table entry (LTE).

The global lock manager knows the names of all the resources hashing to the lock table entry for which it is the GLM. This enables the global lock manager of the LTE to resolve or confirm contentions on behalf of other XES components hashing to the same lock table entry. The global lock manager must handle all subsequent requests from other XESs until the lock table entry is released. When there is no longer an exclusive interest in the lock table entry, the global lock manager releases it.

### 10.10.2 Real contention

Real contention is caused by normal IRLM lock incompatibility between two members. For example, two transactions try to update the same resource at the same time. Real contention requires the most time to resolve. The waiter is suspended as long as the requested lock is not available, and may eventually time out if the timeout interval has expired, or be selected as the victim of a deadlock. Real contention is just an extension of non-data sharing lock suspensions where the holder and waiter are on a different DB2 member of the data sharing group.

OMEGAMON PE reports real contentions as IRLM contentions.

### 10.10.3 False contention

When a transaction requests a lock, the requesting DB2 member calculates a hash value, which is used to calculate to which lock table entry a resource will be assigned in the lock table on the coupling facility. This hash value is calculated by using the name and other relevant information that uniquely defines the resource to be locked. Obviously, the hash value for a given resource is the same from all DB2 members, that is, different DB2's requesting (incompatible) locks on the same resource will hash to same hash class.

The hash value is a 32-bit value passed through IRLM to XES. XES then calculates the hash value based on the maximum number of available lock table entries to assign the resource to a particular lock table entry. This assignment registers a members interest in a particular lock table entry.

It is possible that resources with different names hash to the same lock table entry. This can be the case because the number of LTEs is too small compared to the number of different resource names that can acquire locks, or sometimes because the hash algorithm is not perfect.

When different resources with incompatible locks hash to the same entry in the lock table, this is called false contention, because there cannot be a real conflict, as the lock requests are for different resources.

When XES determines that false contention occurred, it can resolve the contention, and informs the IRLMs to proceed with granting the locks.

But because potential contention has been identified, an XES has to be nominated to become the global lock manager for this LTE. The global manager nominated is the XES that owns an "exclusive" interest in the global byte of the LTE in the lock structure. The other XESs in the group are instructed to send the lock information that they hold pertaining to this lock table entry to the global lock manager XES so that it can determine whether the contention is real or false. Resolution is possible, as each XES holds information about locks that its IRLM has passed to it. This information includes the resource name (RN), hash value (HV), and lock table entry (LTE) position.

Chapter 2, “Planning for data sharing”, in *DB2 Version 9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845 describes the false contention counters.

Figure 10-19 illustrates the false contention processing.

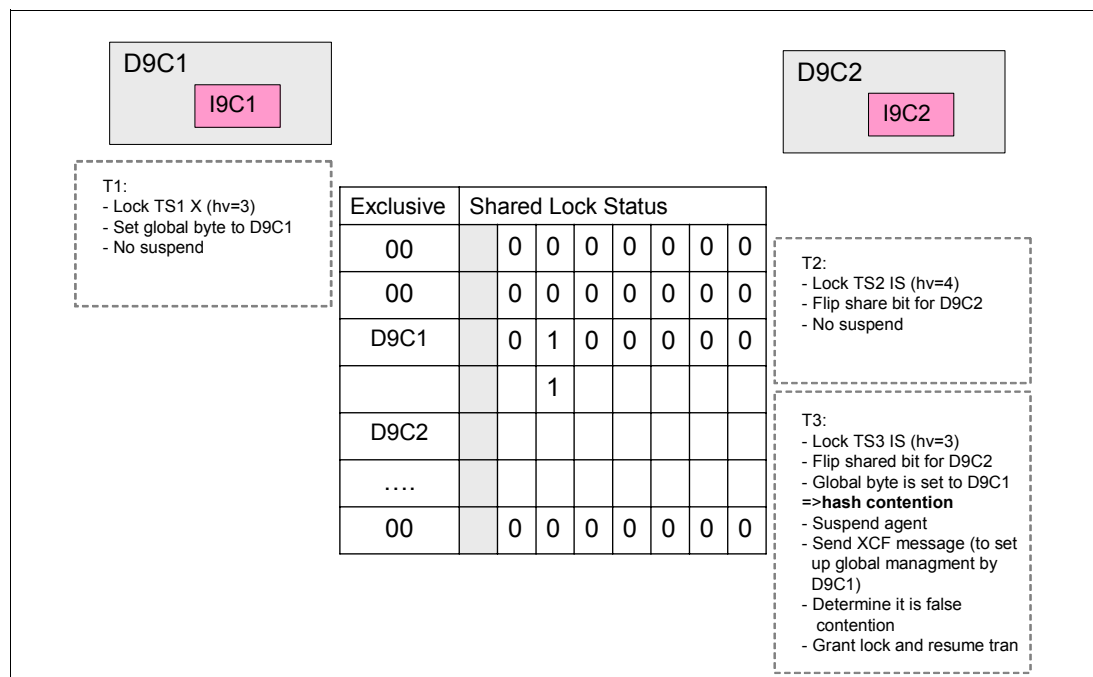


Figure 10-19 False contention resolution

In this example, we use two DB2 members, D9C1 and D9C2. The following set of events occurs in time sequence order:

1. Transaction T1 on D9C1 requests an exclusive X lock on table space TS1. The hash value for table space TS1 is 3. DB2 issues the lock request to IRLM. IRLM determines that the lock needs to be propagated to XES. IRLM issues a “exclusive” lock request to XES for TS1 passing a hash value of 3. XES issues a request to the coupling facility and sets the global byte for hash class 3 to indicate that D9C1 has X type interest. The lock is granted synchronously to transaction T1 and there is no suspension (unless heuristic conversion occurs).
2. Transaction T2 on member D9C2 requests an IS lock on table space TS2. The hash value for TS2 is 4. DB2 issues the lock request to IRLM. IRLM determines that the lock needs to be propagated to XES. IRLM issues a “share” lock request to XES for TS2 passing a hash value of 4. XES issues a request to the coupling facility and sets D9C2’s share bit in hash class 4 to indicate that D9C2 has S-type interest. The lock is synchronously granted to transaction T2 and there is no suspension (unless heuristic conversion occurs). (We assume that member D9C2 is associated with the second connector and therefore will update bit 2 in the shared bit mask. Note that bit zero of the shared bit mask is not used.)

3. Transaction T3 on D9C2 requests an IS lock on table space TS3. The hash value for TS3 is 3. DB2 issues the lock request to IRLM. IRLM determines that the lock needs to be propagated to XES. IRLM issues a “share” lock request to XES for TS3 passing a hash value of 3. XES issues a request to the coupling facility to set D9C2’s share bit in hash class 3, but the coupling facility gives a “hash contention” return code. Transaction T3 is suspended, and XES engages in inter-system XCF messaging to set up global management for hash class 3. In this case, the XES corresponding with D9C1 becomes the global manager for the hash entry. XES gathers up all the locked resource information for hash class 3 from the other systems involved (in this case, the info about the S lock on TS3 is passed to the XES that is associated with D9C1 by way of XCF). Once all the information about all resources that use hash class 3 is gathered, XES that is a GLM can determine that there is no real resource contention (but false contention) in this case (as TS1 and TS3 are different resource names), so XES grants the lock, and transaction T3 is resumed.

#### 10.10.4 XES contention

XES contentions are contentions that can only be resolved by IRLM. XES only has two lock modes (S and X), and IRLM has many more lock modes. For example, IRLM page U and S locks are compatible, but XES sees them as a conflict because an IRLM page U lock is represented by an XES X lock, and an IRLM page S lock is represented by an XES S lock. If XES determines that there is contention on a hash entry, and it is not false contention, it checks whether it is XES contention.

The IRLM page U lock is passed to XES as an X lock. XES tries to register an X mode lock, and an S mode interest is already registered by another member (from our page S lock), it has to check with the IRLMs to determine exactly the types of locks they really are, as it could turn out that according to the IRLMs they are actually compatible. XES passes control to the IRLM contention exit to resolve the contention. The exit checks the actual mode of each IRLM lock, in our case, U and S. If the contention is not real, it is called XES contention, the requested lock can be granted, and the transaction is resumed.

Figure 10-20 shows another example of XES-level resource contention. XES, with its limited “exclusive” and “share” lock state information, detects contention on a resource, and IRLM, with its more detailed lock compatibility states, determines that the contention really does not exist.

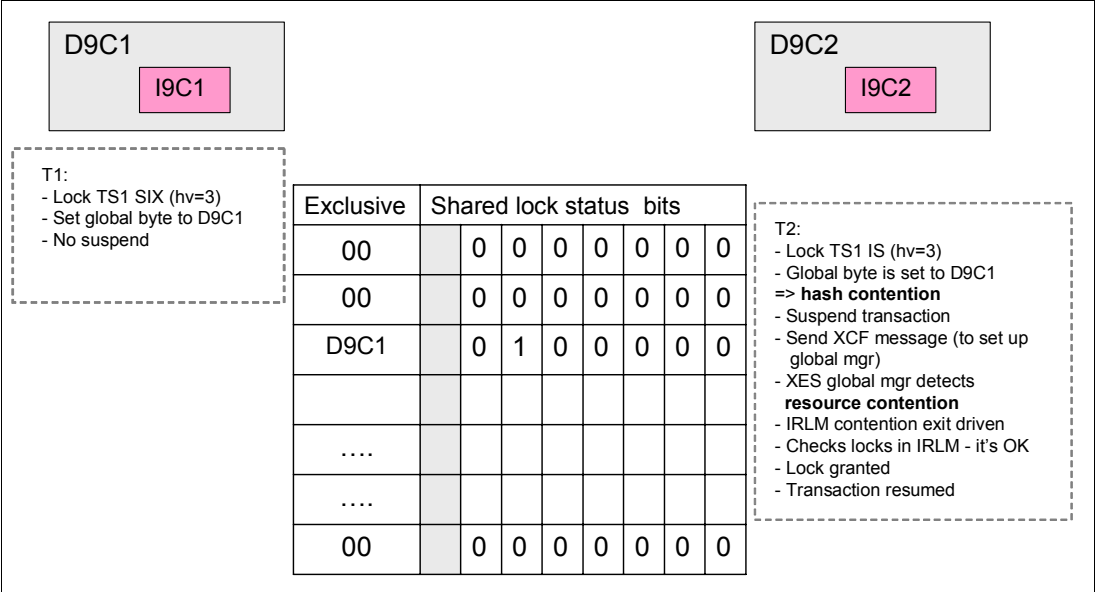


Figure 10-20 XES level resource contention

In the example, the following events happen in time sequence order:

- Transaction T1 on DB2 member D9C1 requests an SIX lock on table space TS1. The hash value for TS1 is 3. DB2 issues the lock request to IRLM. IRLM determines that the lock needs to be propagated to XES. IRLM issues a “exclusive” lock request to XES for TS1, passing a hash value of 3. XES issues a request to the coupling facility and sets the global byte for hash class 3 to indicate that D9C1 has X type interest. The lock is synchronously granted to T1 and there is no suspension (provided no heuristic conversion occurs).
- Transaction T2 on DB2 member D9C2 requests an IS lock on same table space TS1. DB2 issues the lock request to IRLM. IRLM determines that the lock needs to be propagated to XES. IRLM issues a “share” lock request to XES for TS1, passing a hash value of 3. XES issues a request to the coupling facility to set D9C2's share bit, but the coupling facility gives a “hash contention” return code. Transaction T2 is suspended, and XES engages in inter-system XCF messaging to set up global management for hash class 3. In this case, the XES corresponding to D9C1 becomes the global lock manager. XES gathers up all the locked resource information for hash class 3 from all members and determines that there is resource contention between D9C1 and D9C2 (XES sees that D9C1 holds “X” on TS1, and D9C2 wants “S” on TS1). XES drives the IRLM contention exit by way of an SRB scheduled to the IRLM address space, passing all the resource information and the associated “IRLM user data”, which includes the real IRLM lock states. The IRLM contention exit examines this information and determines that the lock states (“SIX” and “IS”) are compatible. IRLM returns “grant” to XES, and XES in turn uses XCF messaging to get the word back to D9C2 that the lock can be granted, and transaction T2 is resumed.

For more details, refer to Chapter 2, “DB2 statistics XES level contention”, in *DB2 Version 9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845.

## 10.10.5 Managing contention

Global contention in a data sharing environment is monitored through the use of the lock table in the coupling facility and through information stored locally in the XES components and IRLMs.

Let us look at how contention is managed in a data sharing environment by stepping through what happens to a lock request that is sent to XES from IRLM (Figure 10-21) and how contention is resolved (Figure 10-22 on page 359).

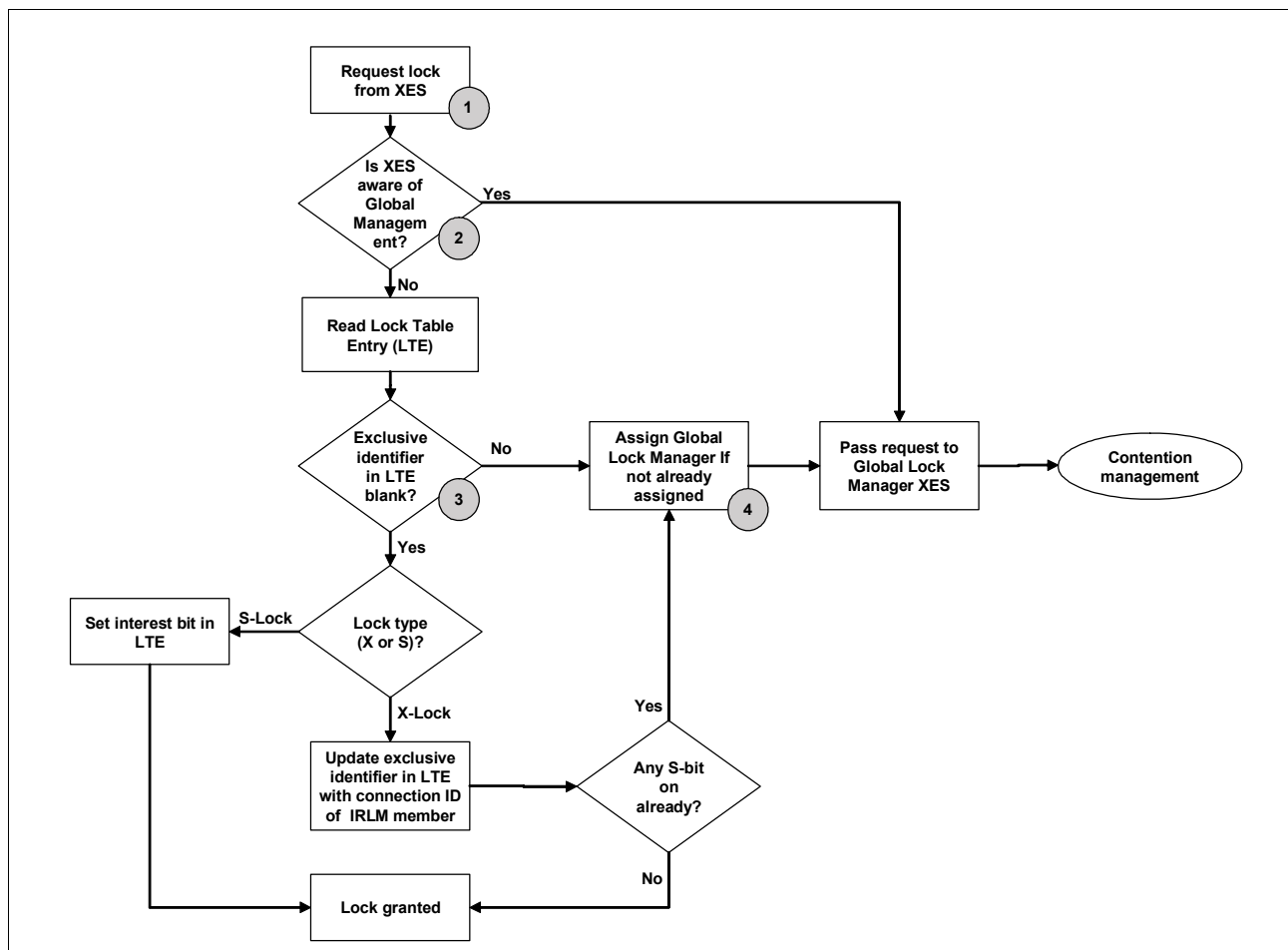


Figure 10-21 Global lock management

Here is what happens when a lock request is sent to XES and contention is detected:

1. When a lock is needed on a resource, DB2 calculates a hash value for the resource to be locked. This 32-bit hash value is calculated using information that uniquely defines the resource to be locked. The hash value and the resource name are passed through IRLM to XES.
2. When the request gets to XES, it hashes the value to calculate to which lock table entry the resource is to be assigned. If global lock management has already been initiated for the lock table entry, this member's XES component will be aware of the fact, if it had an interest in the lock table entry at the time global lock management was initiated. In this case, global lock management is immediately invoked, and there is no communication with the lock table.



3. If there is a member registered as having X mode interest in the global byte of the lock table, global lock management is initiated.
4. Assigning the global lock manager: The XES component of the member that was registered first in the lock table entry as having X mode interest in the lock table entry is appointed as the global lock manager. When global lock management is initiated, all XESs that have an interest in the lock table entry are notified to send their information pertaining to the lock table entry to the global lock manager XES. All XESs that have an interest in the lock table entry store the fact that global lock management is now in place for the lock table entry and store which XES is the global lock manager. Any further lock requests for the lock table entry do not go to the lock table; they are sent directly to the global lock manager XES. New lock requests coming from members that did not have any interest registered for the lock table entry at the time global lock management was initiated will not be aware that global lock management is in place and will go to the lock table before communicating with the global lock manager XES.

Figure 10-22 illustrates the contention management process.

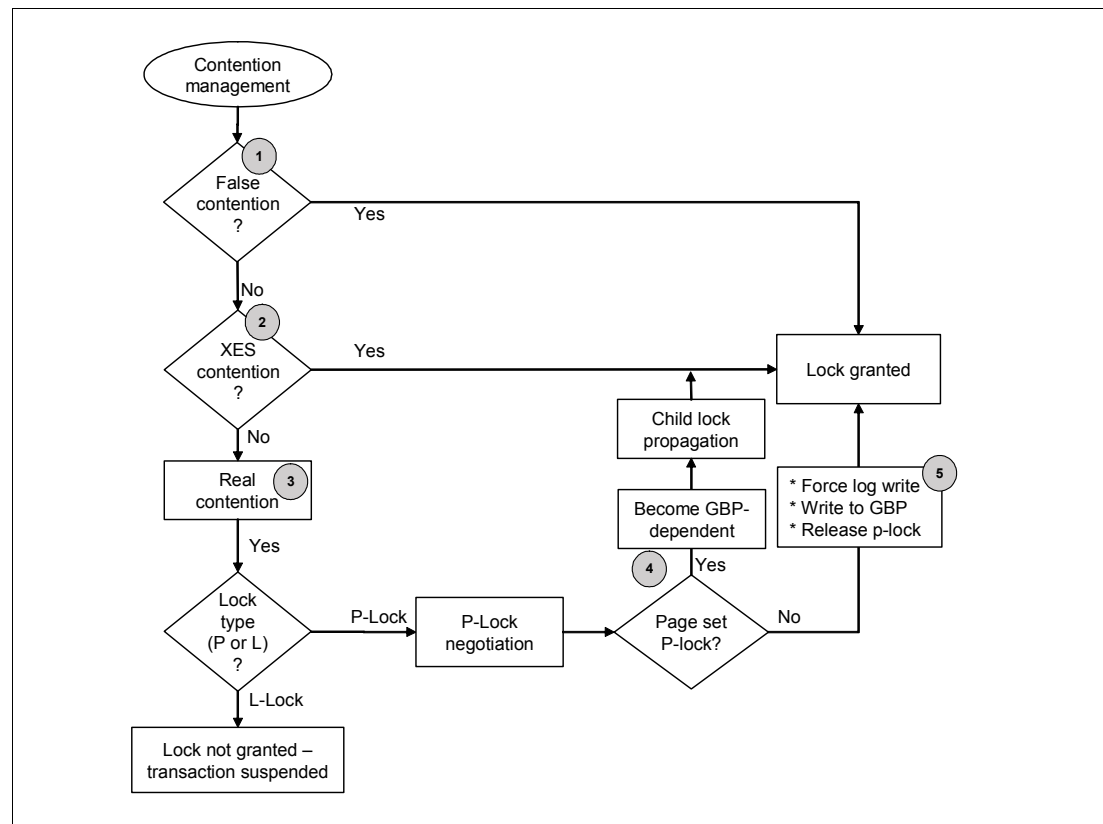


Figure 10-22 Contention management

Where:

1. When contention is detected, the XES that is the GLM for the lock table entry first checks whether false contention exists. XES can do that without the help of IRLM. This process is described in more detail in 10.10.3, “False contention” on page 354.
2. If the contention is not false contention, IRLM is involved to check whether the contention is just XES contention and not real contention. The XES contention resolution process is described in more detail in 10.10.4, “XES contention” on page 356.

3. If the contention is not false contention or XES contention, it must be real contention, where the same resource is involved and the lock requests are not compatible according to the IRLM compatibility matrixes. In that case, we are dealing with real global contention.

Once DB2 has determined that real contention occurs, it checks whether the lock is a P-lock or an L-lock request, as P-locks are negotiable.

4. If the lock is a page set P-lock, IRLM will trigger child lock propagation to the lock structure for child L-locks below the parent L-lock. The negotiation process of the page set P-lock also marks the start of the GBP-dependency for the page set or partition.

Remember that once global lock management has been initiated for a lock table entry, the information stored in the entry is no longer sufficient to grant a lock. Therefore, the only purpose the lock table entry serves at this time is to indicate that global lock management is in place for the lock table entry. All lock requests are directed to the global lock manager XES to resolve, using the information that it maintains about the resources hashed to the lock table entry. Once global lock management is established, it is not de-escalated until the last X-type lock is released for that lock table entry. After de-escalation, members again start using the lock table to register their interest.

5. If the resource that has to be negotiated is a page P-lock, DB2 may have to force a write to the log (log write ahead) and write the page to the GBP before it can release the page P-lock, and then the page P-lock can be granted to the other member.

## 10.11 Timeout and deadlock in a data sharing environment

In DB2 data sharing, deadlocks can occur between transactions on different members. The term global deadlock refers to the situation where two or more members are involved in the deadlock. Local deadlock refers to the situation where all of the deadlocked transactions reside on a single member. The same applies to timeouts. A transaction can get timed out because it has been waiting for a lock that is held by a transaction on another member, in which case it is considered a global timeout.

As is the case in non-data sharing, timeout checking in a data sharing group is also part of the deadlock detection cycle processing, but now it is a more elaborate global deadlock detection cycle.

Global deadlock detection requires the participation of all IRLM members in the data sharing group. Each IRLM member has detailed information about the locks that are held by each member and are being waited for by the transactions on its associated DB2 member.

However, to provide global detection and timeout services, each IRLM is informed of all requests that are waiting globally so that the IRLM can provide information about blockers that are owning locks on this local IRLM. The local IRLM also provides information about the resources for which it is waiting.

To exchange this information, the local IRLM members use XCF messages, so each local IRLM has this global information.

To coordinate the deadlock processing and the exchange of information, one IRLM member assumes the role of the *global deadlock manager* (GDM) for the data sharing group. The GDM is normally the IRLM with the lowest ID, but as IRLM members join or leave the group, the global deadlock manager might change.

As indicated above, each IRLM member in the group must participate in the global deadlock detection process. Each IRLM member (including the one designated as the global deadlock manager) has a role of *local deadlock detector*.

The exchange of four XCF messages represent one complete global detection cycle, which usually takes two to four local intervals to complete.

Four XCF messages are required to gather and communicate the latest information from all the local deadlock detectors. The process is as follows:

1. When the local deadlock detection cycle is triggered, each local deadlock detector sends its information about its lock waiters to the global deadlock manager (GDM).
2. The processing of the GDM is staggered in a way so it kicks in half a deadlock detection cycle later (after the local deadlock process was triggered).

The global deadlock manager takes the information from all local deadlock detectors about the waiters and sends messages about all of them to each of the IRLMs in the group. (Because the global deadlock manager is also a local deadlock detector, it receives the same information, although somewhat quicker than the rest of the IRLMs.)

3. At its next local deadlock interval, each local deadlock detector checks the global view of resources that was sent to it by the GLM, and determines if it has blockers for other waiters. It passes that information along to the global deadlock manager with its list of waiters for the global list of resources that it received.
4. During its next cycle, the global deadlock manager, from the information it received from the local deadlock detectors, determines whether a global deadlock or timeout situation exists. If a global deadlock situation exists, DB2 chooses a candidate for the deadlock.

The global deadlock manager also determines if any timeout candidate is blocked by an incompatible waiter or holder and, if so, presents that candidate to the owning IRLM, along with any deadlock candidates belonging to that IRLM (by way of XCF messages).

When DB2 receives this information, it then determines if it should request that IRLM reject any given timeout candidate waiter or not (DB2 can chose to 'increase the value -worth' of the thread so it does not get timed out, so DB2 makes the ultimate decision).

Figure 10-23 shows the global deadlock process.

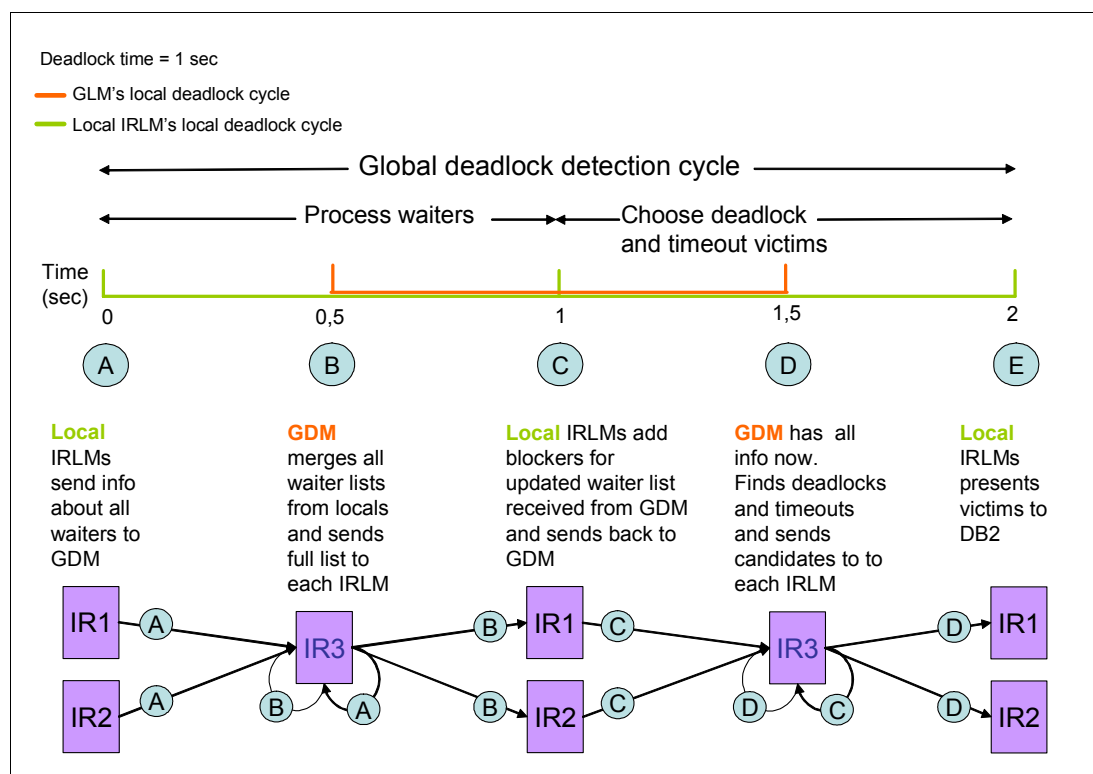


Figure 10-23 Global deadlock detection

Note that deadlock detection can be delayed if any of the IRLMs in the group encounters any of the following conditions:

- ▶ XCF signaling delays. This would delay the exchange of information between the IRLMs, so they cannot continue with the next processing stage, or miss out on some of the information that allows the GDM to detect that a deadlock exists.
- ▶ IRLM latch contention. (This can be encountered in systems with very high IRLM locking activity.) If the deadlock detection process (which runs as a main latch process) cannot run because it has to wait for an IRLM latch, deadlock processing will get delayed.
- ▶ A large number of global waiters. IRLM only allows a limited number of waiters to be presented by an IRLM in a single interval. If there are many waiters, it may take a number of deadlock cycles before they all have been processed, so you can experience longer deadlocks and timeouts that you specified in the DSNZPARMs or IRLM start-up procedure parameters.

## 10.12 Lock avoidance in data sharing

Lock avoidance, discussed in 2.3, “Lock avoidance” on page 33, is important primarily for concurrency, but it is also used for CPU reduction in a data sharing environment. In terms of the CPU, the main saving occurs because IRLM does not propagate the lock request to MVS XES and to the coupling facility.

Lock avoidance uses the same techniques in a data sharing environment as in a non-data-sharing environment.

For non-GBP-dependent page sets or partitions, each member uses the lowest uncommitted log sequence number for all active transactions against that page set or partition, which is exactly how things are in a non-data sharing environment.

However, for GBP-dependent objects, members cannot rely on the local lowest uncommitted log sequence number because other members may have transactions with a lower values for these table spaces. For GBP-dependent page sets, DB2 maintains and uses a value for the entire data sharing group for lock avoidance derived from each member's lowest uncommitted LRSN value. Each member will periodically update its member value and store it in the shared communication area (SCA).

Because lock avoidance for GBP-dependent page sets and partitions is at a member level, even more care has to be taken to ensure effective lock avoidance for GBP-dependent page sets and partitions by committing frequently. For example, one long-running batch job that does not issue frequent commits, even when it only updates non-GBP-dependent objects, will cause fewer pages to qualify for lock avoidance.





## Monitoring data sharing locking activity

In a DB2 data sharing environment, it is very important to monitor the global lock requests propagated to the coupling facility and to control the amount of global contention.

In this chapter, we describe the ways you can monitor the data sharing global locking. We also discuss the possible issues caused by data sharing usage of resources in the coupling facility, such as an undersized lock structure and the overuse of the coupling facility.

The following topics are covered:

- ▶ Monitoring using DB2 commands
- ▶ Monitoring the lock structure size and entries
- ▶ DB2 statistics and accounting traces
- ▶ Monitoring data sharing locks using RMF reports
- ▶ Data sharing global contention scenarios
- ▶ Data sharing lock contention analysis

## 11.1 Monitoring using DB2 commands

We describe three types of commands that can help you monitor global locking in a DB2 data sharing environment:

- ▶ DISPLAY DATABASE
- ▶ DISPLAY BUFFERPOOL
- ▶ ACCESS DATABASE

### **DISPLAY DATABASE**

You can use the -DISPLAY DATABASE LOCKS command to display online information about page set, partition, or table locks that are held on resources. You can also display only those objects that have acquired locks by using the ONLY keyword, for example:

```
-DIS DB(RITA1229) SPACENAM(*) LOCKS ONLY
```

Example 11-1 shows which locks (page set L-locks and P-locks) are being held by all members of the data sharing group.

*Example 11-1 DISPLAY DATABASE SPACE() LOCKS ONLY command output*

---

```
DSNT360I  -D9C1 *****
DSNT361I  -D9C1 *   DISPLAY DATABASE SUMMARY
              *   GLOBAL LOCKS
DSNT360I  -D9C1 *****
DSNT362I  -D9C1      DATABASE = RITA1229  STATUS = RW
              DBD LENGTH = 28256
DSNT397I  -D9C1
NAME      TYPE PART  STATUS              CONNID   CORRID      LOCKINFO
-----
GLWSEMP   TS      0001 RW              TS0       DB2R2       H-IX,P,C
-
-          AGENT TOKEN 5
-          MEMBER NAME D9C2
GLWSEMP   TS      0001 RW              TS0       DB2R2       H-X,PP,I
-
-          MEMBER NAME D9C2
GLWSEMP   TS      0002 RW              TS0       DB2S2       H-S,PP,I
-
-          MEMBER NAME D9C1
GLWSEMP   TS      0002 RW              TS0       DB2S2       H-IS,P,C
-
-          AGENT TOKEN 107
-          MEMBER NAME D9C1
GLWXEMP3  IX      L*    RW              TS0       DB2S2       H-S,PP,I
-
-          MEMBER NAME D9C2
GLWXEMP3  IX      L*    RW              TS0       DB2S2       H-S,PP,I
-
-          MEMBER NAME D9C1
***** DISPLAY OF DATABASE RITA1229  ENDED *****
DSN9022I  -D9C1 DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

---

The application identified as DB2S2 on member D9C1 has a read interest in a page within partition 02 of the GLWSEMP partitioned table space. DB2S2 is running the SQL statement `SELECT ...FROM EMP WHERE EMPNO=hv1`, which means an IS partition L-lock exists on partition 2 (owned by user DB2S2), as well as an S page set P-lock on partition 2 (owned by member D9C1), as D9C1 is the only member reading this partition.

Application DB2R2 on member D9C2 has a write interest in a page of partition 01 and was running the SQL statement `UPDATE EMP .. WHERE EMPNO=hv2`.



As a result, an IX partition L-lock exists on partition 1 (owned by user DB2R2), as well as an page set P-lock on partition 1 (owned by member D9C2), as D9C2 is the only member updating this partition.

Note that partition 1 and 2 are not GBP-dependent. Both partitions are handled independently (partition independence).

Partitions 03 and 04 have no locks and are not shown in the output because the ONLY option was used.

Also note the S page set P-lock on all logical partitions of index GLWXEMP3 for both member D9C1 and D9C2.

Example 11-2 shows the output of the same command after member DC92 experienced a subsystem failure. The L-locks and P-locks are now held in a retained state.

Example 11-2 *DISPLAY DATABASE SPACE() LOCKS ONLY* command output 2

```

DSNT360I  -D9C1 *****
DSNT361I  -D9C1 *   DISPLAY DATABASE SUMMARY
              *   GLOBAL LOCKS
DSNT360I  -D9C1 *****
DSNT362I  -D9C1      DATABASE = RITA1229  STATUS = RW
              DBD LENGTH = 28256
DSNT397I  -D9C1
NAME      TYPE PART  STATUS              CONNID  CORRID      LOCKINFO
-----
GLWSEMP   TS      0001 RW                      R-X,PP
-                MEMBER NAME D9C2
GLWSEMP   TS      0001 RW                      R-IX,P
-                MEMBER NAME D9C2
GLWSEMP   TS      0002 RW                      H-S,PP,I
-                MEMBER NAME D9C1
GLWSEMP   TS      0002 RW      TS0      DB2S2      H-IS,P,C
-                AGENT TOKEN 107
-                MEMBER NAME D9C1
GLWXEMP3  IX      L*   RW                      H-S,PP,I
-                MEMBER NAME D9C1
***** DISPLAY OF DATABASE RITA1229  ENDED *****
DSN9022I  -D9C1 DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***

```

The LOCKINFO field shows whether the lock is retained (R), whether it is held (H), or if we are waiting for the lock (W). For more information about the information in the LOCKINFO field, refer to 9.1.1, “DB2 commands” on page 256.

### **DISPLAY BUFFERPOOL command with GBPDEP(Y)**

You can use the -DISPLAY BUFFERPOOL command with the GBPDEP(Y) keyword to discover if a page set is group buffer pool dependent (GBP-dependent).

Example 11-3 shows the GBP-dependency in page sets for a simple inter-DB2 Write/Write interest scenario. Two transactions, one on each member, update pages in partition 01 using the following SQL statement:

```
UPDATE EMP SET <col>=... WHERE EMPNO = :hv1
```

*Example 11-3 DISPLAY BUFFERPOOL command with GBPDEP(Y)*

---

```
-DIS BUFFERPOOL(BP2) DBNAME(RITA1229) SPACE(*) GBPDEP(YES)
```

```
DSNB401I  -D9C1 BUFFERPOOL NAME BP2, BUFFERPOOL ID 2, USE COUNT 26
...
DSNB460I  -D9C1
-----PAGE SET/PARTITION LIST INFORMATION-----
                        -----DATA SHARING INFO-----
                TS  GBP  MEMBER  CASTOUT  USE  P-LOCK
DATABASE SPACE NAME INST PART IX  DEP   NAME    OWNER   COUNT STATE
=====  =====  ==  ==  ==  ==  ==  ==  ==  ==  ==
RITA1229 GLWSEMP    0001 0001 TS   Y   D9C1      Y       0   IX
                        D9C2      0   IX
DSN9022I  -D9C1 DSNB1CMD '-DIS BUFFERPOOL' NORMAL COMPLETION
```

```
-DIS BUFFERPOOL(BP3) DBNAME(RITA1229) SPACE(*) GBPDEP(YES)
```

```
DSNB401I  -D9C1 BUFFERPOOL NAME BP3, BUFFERPOOL ID 3, USE COUNT 53
...
DSNB460I  -D9C1
-----PAGE SET/PARTITION LIST INFORMATION-----
                        -----DATA SHARING INFO-----
                TS  GBP  MEMBER  CASTOUT  USE  P-LOCK
DATABASE SPACE NAME INST PART IX  DEP   NAME    OWNER   COUNT STATE
=====  =====  ==  ==  ==  ==  ==  ==  ==  ==
RITA1229 GLWXEMP1   0001      IX  Y   D9C1      Y       0   IX
                        D9C2      0   IX
DSN9022I  -D9C1 DSNB1CMD '-DIS BUFFERPOOL' NORMAL COMPLETION
```

---

The top half of the output shows buffer pool BP2 is used by table space GLWSEMP. It also shows the page set P-lock state or mode (intent exclusive (IX) for the partition 01).

The bottom part of the example shows that the SQL UPDATE statements also affected index space GLWXEMP1, which also became GBP-dependent. The display output for buffer pool BP3, which is only used by indexes, shows that the index space also has a page set P-lock of intent to exclusive on both DB2 members.

A good usage scenario for this command is to use it whenever a large batch chain is about to start and you want to check that the page sets are no longer GBP-dependent so that you can avoid the processing impact of lock propagation to the coupling facility and the registering and writing of the pages to the group buffer pool for cache coherency.

## ACCESS DATABASE command

If you discover a number of objects that are still GBP-dependent before running a large batch process, you can remove group buffer pool dependency for a table space, index space, or partition using the -ACCESS DATABASE MODE(NGBPDEP) command. This command was introduced in DB2 9 for z/OS. An example of the command is:

```
-ACCESS DATABASE(RITA1229) SPACE(GLWSEMP) MODE(NGBPDEP)
```

When this command is run, DB2 performs the following actions:

1. Drains all readers and writers on all members other than that on which the command is entered.
2. If step 1) is successful, then the writers on the system on which the command was entered are drained, assuming that the object is not already pseudo-closed.

If the drains fail, the following message is issued:

```
DSNI048I mod-name CLAIMERS EXIST FOR DATA BASE dbname, SPACE NAME tsname, PART  
partno. GROUP BUFFERPOOL DEPENDENCY CANNOT BE REMOVED.
```

STARTDB authority, either explicit or implicit, is required for the -ACCESS command.

The -DISPLAY DATABASE CLAIMERS command can be used to identify what is blocking the drains. If the drains are successful, then the member on which the command was issued converts the object to non-group buffer pool dependent, including the castout of any changed pages. The object is immediately eligible to become group buffer pool dependent again, should there be subsequent access from the other members.

In addition to table spaces, the command is also valid for index spaces.

Note that there is no equivalent command to make an object GBP-dependent. However, you can use the -ACCESS DB command with the MODE(OPEN) option to physically open a page set, partition or index space. This way, you can avoid the situation where the first transaction that accessed the object suffers the processing impact of having to physically open the page set.

**Tip:** In DB2 9 for z/OS, the -ACCESS DB command runs serially, even when multiple jobs issue the command in parallel. Its execution is serialized inside the DB2 engine. Therefore, if you want to open a large number of page sets in parallel, using a simple SELECT 1 FROM *tab* FETCH FIRST 1 ROW ONLY command is recommended.

## 11.2 Monitoring the lock structure size and entries

In this section, we show you how to monitor the lock structure size and discuss ways to change the structure size. It also discusses some of the availability issues regarding the locking structure. See 10.9, “Managing the DB2 lock structure” on page 341, where we explain about how to calculate the size of the lock structure components.

## 11.2.1 The -DISPLAY GROUP command

You can issue the -DISPLAY GROUP command from any active DB2 data sharing member. It displays the coupling facility lock structure size and the number of entries for both the lock list table and record list table, as shown in Example 11-4. For the record list table (also known as the modified resource list), it shows the number of available and in use list entries.

*Example 11-4 DISPLAY GROUP command output*

---

```
DSN7100I  -D9C1 DSN7GCMD
*** BEGIN DISPLAY OF GROUP(DB9CG  ) GROUP LEVEL(910) MODE(N )
          [A]PROTOCOL LEVEL(2)  GROUP ATTACH NAME(D9CG)
-----
```

DB2 MEMBER	ID	SUBSYS	CMDPREF	STATUS	DB2 SYSTEM LVL NAME	IRLM SUBSYS IRLMPROC
D9C1	1	D9C1	-D9C1	ACTIVE	910 SC63	I9C1 D9C1IRLM
D9C2	2	D9C2	-D9C2	ACTIVE	910 SC64	I9C2 D9C2IRLM

---

```
SCA  STRUCTURE SIZE:      8192 KB, STATUS= AC,   SCA IN USE:      4 %
LOCK1 STRUCTURE SIZE:      8192 KB
NUMBER LOCK ENTRIES:      2097152
NUMBER LIST ENTRIES:      11111, LIST ENTRIES IN USE:      8
*** END DISPLAY OF GROUP(DB9CG  )
DSN9022I  -D9C1 DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION
```

---

The output of the -DIS GROUP command also shows that the system is using locking protocol 2, which is what we expected, as this is a V9 NFM system. The output of the display also shows the group attach and the group name.

## 11.2.2 z/OS command D XCF

The z/OS command D XCF,STR,STRNM=struct-nam can be used to display the lock structure size. The D XCF command shows information about the coupling facility, the CFRM policy, and the structures information:

- ▶ CFRM policy definition
- ▶ Preference list
- ▶ Coupling facility name
- ▶ Connections
- ▶ System-managed status

Example 11-5 shows the output of a display of our DB2 data sharing lock structure.

*Example 11-5 z/OS D XCF command*

---

```
D XCF,STR,STRNM=DB9CG_LOCK1

IXC360I 03.49.41 DISPLAY XCF
STRNAME: DB9CG_LOCK1
STATUS: ALLOCATED
EVENT MANAGEMENT: POLICY-BASED
TYPE: LOCK
POLICY INFORMATION:
  POLICY SIZE      : 16384 K
  POLICY INITSIZE: 8192 K
  POLICY MINSIZE  : 0 K
  FULLTHRESHOLD   : 80
  ALLOWAUTOALT    : NO
  REBUILD PERCENT: 5
  DUPLEX          : DISABLED
  ALLOWREALLOCATE: YES
  PREFERENCE LIST: CF1      CF2
  ENFORCEORDER    : NO
  EXCLUSION LIST  IS EMPTY

ACTIVE STRUCTURE
-----
ALLOCATION TIME: 03/08/2009 21:21:50
CFNAME        : CF1
COUPLING FACILITY: 002094.IBM.02.00000002991E
                PARTITION: OF  CPCID: 00
ACTUAL SIZE    : 8192 K
STORAGE INCREMENT SIZE: 512 K
USAGE INFO     TOTAL      CHANGED   %
ENTRIES:       11111      11        0
LOCKS:         2097152
...

```

---

In Example 11-5, we can see the options used by the lock structure DB9CG\_LOCK1 in the CFRM policy, the lock structure size, and all sorts of information related to the lock structure coupling facility usage. In 10.9.1, “Definition of the lock structure” on page 341, we provide information about the lock structure options.

### ***Lock structure duplexing***

All CF structures that are used by DB2 can be duplexed. This can help improve availability, as a second copy is immediately available in case the primary copy breaks or the systems lose connection to it. However, there are performance implications of which you should be aware.

To duplex DB2 group buffer pools, DB2 uses so-called *user-managed duplexing*. This type of duplexing is implemented by the exploiter (DB2) of the CF structure (the group buffer pool). DB2’s duplexing implementation for group buffer pools is very efficient. The performance impact is small. Since recovering from a loss of the group buffer pool is very disruptive, duplexing DB2 group buffer pools is highly recommended.

Duplexing of the DB2 SCA and the IRLM lock structure is done by way of *system-managed duplexing*. System-managed duplexing is implemented by XES and is applicable to other (non-DB2) structures in the coupling facilities. System-managed duplexing applies to lock and SCA structures, not to group buffer pools.

System-managed duplexing uses a different technique to keep both copies of the structure synchronized than DB2 uses to duplex the group buffer pools. Duplexing the SCA and lock structure can have a significant performance impact, especially when the distance between CFs, or between the CF and the LPAR, increases. Also, because the SCA and lock structure can be rebuilt faster than a group buffer pool, duplexing them normally does not increase availability to the same extent as duplexing the group buffer pools does.

Roughly speaking, a lock request for a duplexed lock structure can easily take three times the cost of a simplexed DB2 lock request. For example, if a synchronous lock takes 10  $\mu$ sec., then a duplexed lock, when performed synchronously, would take about 30  $\mu$ sec. However, it is likely that the lock request is converted to asynchronous in order to save host processor CPU cycles, which must spin while synchronous activity is performed. The asynchronous lock is likely to take more than 100  $\mu$ sec. at the least.

For more information, refer to *System-Managed Coupling Facility Structure Duplexing*, GM13-0103, which can be found at the following address:

<http://www.ibm.com/servers/eserver/zseries/library/techpapers/gm130103.html>

## 11.3 DB2 statistics and accounting traces

Periodic monitoring with DB2 statistics and accounting traces is important in a data sharing environment for monitoring the global locking activity. Section 9.1.3, “Standard DB2 traces” on page 265 has information regarding the L-locks that both DB2 non-data sharing and data-sharing use.

In this section, we focus on the counters that are related to DB2 global locking.

The amount of information gathered for the analysis for your DB2 environment depends on the level of DB2 trace data collected.

As indicated earlier in this publication, collecting DB2 statistics traces records for classes 1, 3, 4, and 5 with a statistics interval of one minute (DSNZPARM STATIME=1) is highly recommended. Also, collecting DB2 accounting trace for classes 1, 2, 3, 7, and 8 is usually not a problem for most customers.

For a more detailed description of these traces, the data they collect, and the processing impact associated with them, refer to 8.3.1, “Which information to gather” on page 227.

In a DB2 data sharing environment, by default, DB2 traces are taken at a member level and must be run on all DB2 members that are being monitored. When you can start collecting DB2 traces in data sharing, you can either use the following options:

- ▶ SCOPE(LOCAL): The default; the traces are collected only in the member where you start the trace.
- ▶ SCOPE(GROUP): Starts the traces on all the (active) data sharing members.

### 11.3.1 OMEGAMON PE SCOPE(MEMBER/GROUP) reports

The OMEGAMON XE for DB2 Performance Expert statistics and accounting *reports* also have a SCOPE subcommand, which has the following options:

- ▶ **SCOPE(MEMBER);** The default; shows the all statistics identifiers report at a member level.
- ▶ **SCOPE(GROUP);** It combines the DB2 trace data of the individual members and presents the data sharing activity and group buffer pool activity for the entire group. To generate these group-scope reports, you have to provide the individual member's trace data sets as input.

The OMEGAMON PE SCOPE subcommand is not related to the SCOPE keyword on the -START TRACE command at all. The OMEGAMON PE SCOPE subcommand only indicates how the reports are to be generated, and is not dependent on the SCOPE option that was used when the DB2 trace data was collected.

Example 11-7 on page 375 shows the layout of an OMEGAMON PE STATISTICS REPORT SCOPE(GROUP).

*Example 11-6 Layout of the STATISTICS SCOPE(GROUP) report*

---

```
LOCATION: DB9C                OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)                PAGE: 1-1
GROUP: DB9CG                STATISTICS REPORT - LONG                                REQUESTED FROM: NOT SPECIFIED
                                                                    TO: NOT SPECIFIED
                                                                    INTERVAL FROM: 08/16/09 19:18:23.76
                                                                    TO: 08/16/09 19:23:44.35

DB2 VERSION: V9                SCOPE: GROUP

---- HIGHLIGHTS -----
INTERVAL START : 08/16/09 19:18:23.76  SAMPLING START: 08/16/09 19:18:23.76  TOTAL THREADS   :    1.00
INTERVAL END   : 08/16/09 19:23:23.76  SAMPLING END   : 08/16/09 19:23:23.76  TOTAL COMMITS   :   661.00
INTERVAL ELAPSED:    5:00.000148      OUTAGE ELAPSED:    0.000000      DATA SHARING MEMBER: D9C1

DATA SHARING LOCKING          QUANTITY /SECOND /THREAD /COMMIT
-----
...
GROUP BPO                      QUANTITY /SECOND /THREAD /COMMIT  GROUP BPO  CONTINUED  QUANTITY /SECOND /THREAD /COMMIT
-----
...
LOCATION: DB9C                OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)                PAGE: 1-9
GROUP: DB9CG                STATISTICS REPORT - LONG                                REQUESTED FROM: NOT SPECIFIED
                                                                    TO: NOT SPECIFIED
                                                                    INTERVAL FROM: 08/16/09 19:18:23.76
                                                                    TO: 08/16/09 19:23:44.35

DB2 VERSION: V9                SCOPE: GROUP

---- HIGHLIGHTS -----
INTERVAL START : 08/16/09 19:18:44.35  SAMPLING START: 08/16/09 19:18:44.35  TOTAL THREADS   :    1.00
INTERVAL END   : 08/16/09 19:23:44.35  SAMPLING END   : 08/16/09 19:23:44.35  TOTAL COMMITS   :   661.00
INTERVAL ELAPSED:    5:00.000051      OUTAGE ELAPSED:    0.000000      DATA SHARING MEMBER: D9C2

...

LOCATION: DB9C                OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)                PAGE: 1-17
GROUP: DB9CG                STATISTICS REPORT - LONG                                REQUESTED FROM: NOT SPECIFIED
                                                                    TO: NOT SPECIFIED
                                                                    INTERVAL FROM: 08/16/09 19:18:23.76
                                                                    TO: 08/16/09 19:23:44.35

DB2 VERSION: V9                SCOPE: GROUP

---- HIGHLIGHTS -----
INTERVAL START : 08/16/09 19:18:23.76  SAMPLING START: 08/16/09 19:18:23.76  TOTAL THREADS   :    2.00
INTERVAL END   : 08/16/09 19:23:44.35  SAMPLING END   : 08/16/09 19:23:44.35  TOTAL COMMITS   :  1322.00
INTERVAL ELAPSED:    5:00.000099      OUTAGE ELAPSED:    0.000000      DATA SHARING MEMBER:    2

...
...

```

---

Note the following about the group scope statistics report:

- ▶ The DB2 statistics intervals (INTERVAL START) are not synchronized in the group by default. This can sometimes make it difficult to interpret the data, especially when the DB2 statistics interval is long. You can have the DB2 systems synchronize their statistics interval more closely by way of the SYNCVAL DSNZPARM. The default is NO, which means no synchronization. You can specify a value of 0 to 59 to synchronize with the clock time, where 0 means on the hour. This DSNZPARM is also used to synchronize the DB2 statistics with the RMF interval.
- ▶ The global scope reports always contain the data for each of the members first, and then the totals for the group. For the statistics report, this is indicated by "DATA SHARING MEMBER: #", where # is the number of members in the data sharing group that is being reported. In our case, there are two active members in the data sharing group. In the global scope accounting reports, the group totals are indicated by the "\*\*\* GRAND TOTAL \*\*\*" heading.
- ▶ For the statistics report, it only contains the data sharing locking and the group buffer pool sections, not any of the other sections that you normally expect in an OMEGAMON PE statistics report. The accounting global scope report contains all counters of a normal accounting report (not just global locking and group buffer pool data).
- ▶ When you look at the counters for an individual member, they are the number of requests from that member. For example, LOCK REQUESTS (P-LOCKS) 3456 means that this member issued 3456 P-lock requests in the interval you are reporting. Only in the last part of the reports, that report on the entire data sharing group, are the values of all members combined.

Note that our workload environment was only used with the purpose of showing some insights about the DB2 data sharing global locking behavior, and is not really a reflection of a real world workload.

### 11.3.2 DB2 statistics data

As indicated before, the DB2 statistics trace runs with very low processing impact and should be turned on permanently for each member of the data sharing group to allow continuous monitoring of each DB2 data sharing member, as well as the group.

The LOCKING ACTIVITY section of the DB2 statistics report shows information about all the L-lock activity of the data sharing member. The LOCKING ACTIVITY section is only present in the SCOPE(MEMBER) statistics report. For more information about the counters in this section, refer to Part 3, "Monitoring and problem determination" on page 217.

Both the SCOPE(GROUP) and SCOPE(MEMBER) STATISTICS REPORT or TRACE have a DATA SHARING LOCKING section.



## Data sharing locking section

Example 11-7 shows the data sharing activity section of the DB2 statistics report for the entire DB2 data sharing group DB9CG (DATA SHARING MEMBER: 2).

### Example 11-7 Statistics data sharing locking

---- HIGHLIGHTS				
-----				
INTERVAL START	: 08/16/09 19:18:23.76	SAMPLING START:	08/16/09 19:18:23.76	
TOTAL THREADS	: 2.00			
INTERVAL END	: 08/16/09 19:23:44.35	SAMPLING END	: 08/16/09 19:23:44.35	
TOTAL COMMITS	: 1322.00			
INTERVAL ELAPSED:	5:00.000099	OUTAGE ELAPSED:	0.000000	
<b>DATA SHARING MEMBER: 2</b>				
DATA SHARING LOCKING	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
GLOBAL CONTENTION RATE (%)	4.25			
P/L-LOCKS XES RATE (%)	61.90			
LOCK REQUESTS (P-LOCKS)	28180.00	93.93	14.1K	21.32
UNLOCK REQUESTS (P-LOCKS)	24133.00	80.44	12.1K	18.25
CHANGE REQUESTS (P-LOCKS)	77.00	0.26	38.50	0.06
SYNCH.XES - LOCK REQUESTS	78312.00[A]	261.04	39.2K	59.24
SYNCH.XES - CHANGE REQUESTS	1945.00[B]	6.48	972.50	1.47
SYNCH.XES - UNLOCK REQUESTS	46601.00[C]	155.34	23.3K	35.25
ASYNCH.XES - RESOURCES	11.00[D]	0.04	5.50	0.01
SUSPENDS - IRLM GLOBAL CONT	5044.00[E]	16.81	2522.00	3.82
SUSPENDS - XES GLOBAL CONT.	50.00[F]	0.17	25.00	0.04
SUSPENDS - FALSE CONTENTION	537.00[G]	1.79	268.50	0.41
INCOMPATIBLE RETAINED LOCK	0.00	0.00	0.00	0.00
NOTIFY MESSAGES SENT	301.00	1.00	150.50	0.23
NOTIFY MESSAGES RECEIVED	147.00	0.49	73.50	0.11
P-LOCK/NOTIFY EXITS ENGINES	1000.00	N/A	N/A	N/A
P-LCK/NFY EX.ENGINE UNAVAIL	0.00	0.00	0.00	0.00
PSET/PART P-LCK NEGOTIATION	72.00[H]	0.24	36.00	0.05
PAGE P-LOCK NEGOTIATION	4812.00[I]	16.04	2406.00	3.64
OTHER P-LOCK NEGOTIATION	39.00[J]	0.13	19.50	0.03
P-LOCK CHANGE DURING NEG.	4914.00	16.38	2457.00	3.72

The statistics report taken by the SCOPE(MEMBER) option for each member has the L-locking activity and the data sharing locking activity sections for that member. You should analyze the data sharing systems global locking activity using both sections, the (L-) LOCKING ACTIVITY and DATA SHARING LOCKING sections. (The locking activity section only contains information about L-locks (the ones that are not propagated to XES + the ones sent to XES; P-locks are not included).

Now we look at each of the counters of the data sharing locking section (in the DB2 statistics report) in more detail, and discuss the type of information they contain.

### **LOCK REQUEST (P-LOCKS)**

The number of total lock requests for physical locks (P-locks). This counter includes page set/partition and page P-locks requests. P-lock request are always sent to XES.

There is additional information about page P-locks at the group buffer pool level. Refer to “Group buffer pool section” on page 381 for details.

### **UNLOCK REQUEST (P-LOCKS) / CHANGE REQUEST (P-LOCKS)**

These numbers represent the number of unlock and change requests made for P-locks (both page set and page P-locks). The unlock requests counted here are sent synchronously to XES by IRLM.

### **SYNCH.XES - LOCK REQUEST**

The total number of lock requests that have been synchronously sent to XES. This number includes both P-locks (page set and page) and L-lock requests propagated to z/OS XES synchronously. This number is not incremented if the request is suspended during processing.

### **SYNCH.XES - UNLOCK REQUEST/ SYNCH.XES - CHANGE REQUEST**

These counters represents the number of unlock and change requests (including L-locks and P-locks) made to the coupling facility lock structure. The unlock request to the coupling facility is synchronous, but you have a list of locks in a single unlock request.

Note that in a DB2 data sharing environment, IRLM asks to send the L-lock and P-lock requests synchronously to XES and the coupling facility lock structure. However, z/OS XES may decide to convert the synchronous global locking requests to the coupling facility into asynchronous requests. This is called *heuristic conversion*.

At the time of writing of this publication, synchronous requests that are converted into asynchronous requests by the heuristic conversion algorithm are *not* included in the SYNCH.XES (lock/unlock/change) counters. These only include synchronous requests that were processed synchronously (and were not converted to asynchronous by the heuristic algorithm).

To verify whether you have this problem, you should verify the RMF CF ACTIVITY reports and verify the number of ASYNC REQ for the DB2 lock structure. If the majority of the CF lock requests is ASYNC, it is best to use the RMF reports to calculate the DB2 global and false contention rate (refer to 11.4, “Monitoring data sharing locks using RMF reports” on page 389 for more information).

APARs PK85159 and PK85543, both open at the time of writing, should resolve this problem.

More background information about this issue can be found in the Techdoc *Asynchronous DB2 data sharing locks not counted*, found at:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/FLASH10632>

### **ASYNCH.XES - RESOURCES**

This counter represents the number of child L-locks that were propagated to XES asynchronously when a change in the page set P-lock state triggered child lock propagation. Normally, DB2 data sharing propagates the locks synchronously to the coupling facility lock structure. Asynchronous mean that the request was done under a system execution unit and not in the allied work unit. This counter does not include the number of synchronous requests that were converted to asynchronous by the heuristic algorithm.

The previous set of counters is related to the number of requests that are sent to XES, and that were granted without a “problem”.

Now we look at the number of requests that experienced some form of contention (and had to be suspended to resolve the contention).

### ***SUSPENDS - IRLM GLOBAL CONT***

The number of suspensions due to real global contention, that is, the IRLM lock states on a single resource are in conflict or contention.

### ***SUSPENDS - XES GLOBAL CONT***

This counter represents the number of suspensions due to XES contention. XES only recognizes two lock modes, X and S, and becomes suspended due to XES lock incompatibility. However, IRLM does not have contention with the lock states or modes.

### ***SUSPENDS - IRLM FALSE CONTENTION***

This counter tracks the number times a request was suspended due to two different resources being hashed to same entry in the lock table of the coupling facility lock structure. The IRLM locks modes are not in conflict, as there are two different resources involved.

Because of the difficulties DB2 is currently (at the time of writing of this publication) having distinguishing between a request that suffered from false contention and a request that was only suspended because of heuristic conversion, the number of false contentions (QTGSFLMG) is retrieved directly from the coupling facility. However, this has an interesting side effect of which you should be aware: This false contention counter in the CF is maintained at the LPAR level. Therefore, this counter overreports false contentions when multiple members of the same DB2 data sharing group run on the same z/OS image.

Table 11-1 provides a quick reference of which type of lock requests are included in the different counters of the locking sections on the OMEGAMON PE statistics report.

*Table 11-1 Locking counters overview*

OMEGAMON PE locking counter	Includes								
	L-lock						P-lock		
	Kept local			Sent to XES					
	Page-set	Page row	Other	Page-set	Page / row	Other	Page-set	Page row	Other
<b>LOCKING ACTIVITY</b>									
LOCK REQUESTS	X	X	X	X	X	X			
UNLOCK REQUESTS	X	X	X	X	X	X			
QUERY REQUESTS	X	X	X	X	X	X			
CHANGE REQUESTS	X	X	X	X	X	X			
<b>DATA SHARING LOCKING</b>									
LOCK REQUESTS (P-LOCKS)							X	X	X
UNLOCK REQUESTS (P-LOCKS)							X	X	X
CHANGE REQUESTS (P-LOCKS)							X	X	X
SYNCH.XES - LOCK REQUESTS				X	X	X	X	X	X
SYNCH.XES - CHANGE REQUESTS				X	X	X	X	X	X

OMEGAMON PE locking counter	Includes								
	L-lock						P-lock		
	Kept local			Sent to XES					
	Page-set	Page row	Other	Page-set	Page / row	Other	Page-set	Page row	Other
SYNCH.XES - UNLOCK REQUESTS				X	X	X	X	X	X
ASYNCH.XES - RESOURCES					X				

### GLOBAL CONTENTION RATE(%)

The *global contention rate* indicates the level of contention that occurred within the data sharing group.

The global contention rate is calculated as the total number of suspensions (real (or IRLM) contention, XES contention, and false contention) divided by the total number of requests that were sent to XES, multiplied by 100.

We use information from Example 11-7 on page 375 to calculate the global contention rate as follows:

$$E + F + G / [A + B + C + D + E + F + G] * 100$$

Where:

- ▶ E + F + G is the total of number of suspensions.
- ▶ A + B + C + D + E + F + G is the total number of XES/IRLM requests.

So based on the numbers in our example, we calculate the global contention rate as:

$$(5044 + 50 + 537) / (78312 + 1945 + 46601 + 11 + 5044 + 50 + 537) * 100$$

$$5631 / (126869 + 5631) * 100 = 4.25\%$$

This number is relatively high, but is still a good value. However, at this time, the values of [A], [B], and [C] only include the requests that were sent synchronously to XES (and did not get converted by the heuristic conversion algorithm), which tends to overestimate the global contention rate if a great deal of heuristic conversion occurs. OMEGAMON PE does this calculation for you, so you only have to look at the value for "GLOBAL CONTENTION RATE(%)".

If you experience a high number of XES and false contentions, you should investigate this situation in more detail, as locking protocol 2 should have introduced a significant reduction in both types of contention.

When the global contention rate is low, it is probably not necessary to calculate false contention rate. False contention is calculated in the following way:

$$G / [A + B + C + D + E + F + G] * 100$$

Using the numbers in the example, this leads to:

$$537 / (78312 + 1945 + 46601 + 11 + 5044 + 50 + 537) * 100$$

$$537 / 132500 * 100 = 0.4\% \text{ false contention rate}$$

**Tip:** The ROT for the global contention rate should be less than 3 to 5% of the *total number of XES/IRLM requests*.

If there are significant values in the global contention rate, verify and measure the false contention rate. False contention should be less than 1 to 1.5% of the *total number of XES/IRLM requests*.

In Example 11-7 on page 375, the global contention rate is 4.25%, and therefore within in the range of 3 to 5%.

### **P/L-LOCKS XES RATE(%)**

This value shows the percentage of P-locks and L-lock requests that were propagated to XES synchronously, compared to the total number of lock requests. This number is an indication of the impact of the usage of explicit hierarchical locking and other locking optimizations. In an environment where all the workload is sharing data across the different members, a value of 60% means that 40% of all transaction locks were not propagated to XES due to data sharing locking optimizations.

The P/L-Locks XES RATE is calculated as follows:

$$[\text{SYNCH.XES-LOCK REQUEST}]/([\text{LOCK REQUEST (L-LOCKS)}]+[\text{LOCK REQUEST (P-LOCKS)}])*100$$

In our workload example, the D9C1 and D9C2 statistics report locking activity section has information about the total number of L-lock requests (see Example 11-8):

- ▶ D9C1 (L-)LOCK REQUESTS - 50834.00
- ▶ D9C2 (L-)LOCK REQUESTS - 47502.00

Note that this information is not available in the STATISTICS SCOPE(GROUP) report. For a statistics report, this only contains information about the data sharing locking and group buffer pool activity.

#### **Example 11-8 Statistics LOCKING ACTIVITY section for both members**

---

SUBSYSTEM: D9C1  
DB2 VERSION: V9

LOCKING ACTIVITY	QUANTITY
LOCK REQUESTS	50834.00

SUBSYSTEM: D9C2

LOCKING ACTIVITY	QUANTITY
LOCK REQUESTS	47502.00

---

The data sharing locking activity in Example 11-7 on page 375 has the other counters:

- ▶ LOCK REQUESTS (P-LOCKS): 28180.00
- ▶ SYNCH.XES - LOCK REQUESTS: 78312.00

The calculation is  $78312.00 / 126516.00 * 100 = 61.89\%$ .

See the (L-)locking activity section of the statistics report (described in “Statistics locking activity” on page 230).

### ***INCOMPATIBLE RETAINED LOCK***

This is the number of global lock or change requests denied or suspended due to an incompatible retained lock. This value should be zero.

### ***NOTIFY MESSAGES SENT/NOTIFY MESSAGES RECEIVED***

This is the number of IRLM notify messages sent to and received from other members in the group.

### ***P-LOCK/NOTIFY EXITS ENGINES***

This is the maximum number of engines available for physical lock exit or notify exit requests.

### ***P-LCK/NFY EX.ENGINE UNAVAIL***

This is the number of times an engine is not available for physical lock exit or notify exit requests.

### ***PSET/PART P-LCK NEGOTIATION***

This is the number of times this DB2 was driven to negotiate a partition or page set physical lock due to changing inter-DB2 read/write or inter-DB2 write/write interest levels on the partition or page set.

### ***PAGE P-LOCK NEGOTIATION***

This is the number of times this DB2 was driven to negotiate a page physical lock because of page P-lock contention with another DB2 member. Page P-locks are used when row locking is in effect, but also for space map pages and index leaf pages for indexes regardless of locking level, when the objects are GBP-dependent. Our workload example has considerable sequential INSERTs in the same tables in both DB2 members, which is there is a high number of page P-lock negotiations.

For additional information about the types of page P-locks negotiation at the group buffer pool level, refer to “Group buffer pool section” on page 381.

### ***OTHER P-LOCK NEGOTIATION***

This is the number of times this DB2 was driven to negotiate a physical lock type other than page set, partition, or page. Other P-lock negotiation include index tree P-locks, castout P-locks, and SKCT/SKPT P-locks.

**Tip:** The ROT for the total number of P-lock negotiations should be less than 3 to 5% of the total number of XES/IRLM requests. The total number of P-Lock negotiations is PSET/PART P-LCK NEGOTIATION + PAGE P-LOCK NEGOTIATION + OTHER P-LOCK NEGOTIATION.

In Example 11-7 on page 375, the *total number of P-lock negotiations* is the sum of the counters [H]+[I]+[J], and [A] + [B]+ [C] +[D] + [E] + [F] + [G] is the *total number of XES/IRLM requests*.

In our example we have:

$$(72 + 4812 + 39) / (78312 + 1945 + 46601 + 11 + 5044 + 50 + 537) * 100$$

$$4923 / 132500 * 100 = 3.71\%$$

### ***P-LOCK CHANGE DURING NEG***

This is the number of times a physical lock change request was issued during physical lock negotiation. Remember that DB2 in a data sharing environment performs dynamic tracking of inter-DB2 interest.

### **Group buffer pool section**

This section describes each of the counters related to page P-locks that are tracked at the GBP level. They can be found at the end of the Group Buffer Pool Activity section of the DB2 statistics report.

Example 11-9 shows the group buffer pool counters related to page P-locks. They include a breakdown of page P-locks for space map pages, data pages (when using row level locking), and index leaf pages. In addition to the number of page P-lock requests, the report also shows the number of page P-lock suspensions and negotiations for each page P-lock type.

*Example 11-9 Page P-lock counters in the GBP section*

GROUP TOTAL	CONTINUED	QUANTITY	/SECOND	/THREAD	/COMMIT
-----		-----	-----	-----	-----
.....					
PAGE P-LOCK LOCK REQ		183.4K	611.41	91.7K	138.75
SPACE MAP PAGES		6463.00	21.54	3231.50	4.89
DATA PAGES		130.00	0.43	65.00	0.10
INDEX LEAF PAGES		176.8K[K]	589.44	88.4K	133.76
PAGE P-LOCK UNLOCK REQ		180.5K	601.74	90.3K	136.55
PAGE P-LOCK LOCK SUSP		6001.00	20.00	3000.50	4.54
SPACE MAP PAGES		3400.00[L]	11.33	1700.00	2.57
DATA PAGES		2.00[M]	0.01	1.00	0.00
INDEX LEAF PAGES		2599.00[N]	8.66	1299.50	1.97
PAGE P-LOCK LOCK NEG		5179.00	17.26	2589.50	3.92
SPACE MAP PAGES		3367.00	11.22	1683.50	2.55
DATA PAGES		0.00	0.00	0.00	0.00
INDEX LEAF PAGES		1812.00	6.04	906.00	1.37

In our test scenario, there are a significant number of INSERT (and also UPDATE) statements in each member. The tables also have a significant number of indexes on each table. During each workload, the same number of SQL statements run, and the same SQL DML statements run on both members. The workload is designed to generate a lot of global lock contention.

In Example 11-9, in the GROUP TOTAL for the group buffer pool activity section, we have a high number of PAGE P-LOCK LOCK REQ(183.4K)s and PAGE P-LOCK UNLOCK REQ(180.5K)s for the group, mainly because of page P-lock requests to INDEX LEAF PAGES[K], which have a total of 176.8K requests.

The page P-lock suspensions we have in our example are for SPACE MAP PAGES[L]and INDEX LEAF PAGES[N].

Next, we provide a short description of the OMEGAMON PE DB2 statistics counters related to page P-lock activity from the group buffer pool activity section.

### **Page P-LOCK LOCK REQ**

This is the total number of page P-lock lock requests.

#### ***SPACE MAP PAGES***

This is the number of page P-lock lock requests for space-map pages. You get this value when more than one data sharing member is performing INSERT/UPDATE/DELETE in the page set. Page map updates in a non-data sharing environment do not use (L-)locks to serialize updates; they use page latches instead. In a data sharing environment, these updates are extended to page P-lock requests when the pageset is GBP-dependent.

#### ***DATA PAGES***

This is the number of page P-lock lock requests for data pages. DB2 uses page P-lock requests when the table space data uses LOCKSIZE ROW (row-level locking) and the object is GBP-dependent.

#### ***INDEX LEAF PAGES***

This is the number of page P-lock lock requests for index-leaf pages. This counter is for INSERT/UPDATE activities in an index space page set for more than one member. As there are no locks on the index pages, updates across members for GBP-dependent indexes are serialized by way of a page P-lock. Note that an index can be GBP-dependent while a table space is not, and vice versa. Both are handled independently. In Example 11-9 on page 381, the majority of the page P-lock requests are because of INDEX LEAF PAGES[K].

### **PG P-LOCK UNLOCK REQ**

This is the number of page P-lock unlock requests. Note that the page P-lock unlock requests are requested to be processed synchronously by IRLM.

### **PG P-LOCK LOCK SUSP**

This is the sum of all page P-lock lock suspensions.

#### ***SPACE MAP PAGES***

This is the number of page P-lock suspensions for space-map pages.

#### ***DATA PAGES***

This is the number of page P-lock lock suspensions for data pages.

#### ***INDEX LEAF PAGES***

This is the number of page P-lock lock suspensions for index-leaf pages.

## **11.3.3 DB2 accounting data**

In this section, we describe the different counters that can be found in DB2 accounting records that are specific to data sharing locking.

The OMEGAMON PE DB2 accounting reports (TRACE and REPORT options) show the global locking activity and group buffer pool activity at the application level in a data sharing environment.



Unlike the statistics group-scope report that only contains sections specific to data sharing (data sharing locking and group buffer pool activity), the ACCOUNTING REPORT SCOPE(GROUP) command generates a report per member, and a “grand total” report for the entire group with all the sections and counters that are present in the (default) scope(member) reports.

The accounting reports we use in this section were produced using the following OMEGAMON PE command:

```
ACCOUNTING
REPORT
ORDER(CORRNAME)
SCOPE(GROUP)
LAYOUT (LONG)
```

During our test scenario, we ran two jobs, DB2GER63 and DB2GER64, one on each member. The workload runs a number of stored procedures using the DB2 Workload Generator V1.5 tool. Each job performs the same number and type of SQL statements. This allows us to generate a high degree of global contention in our data sharing environment to illustrate the counters to investigate.

### Data sharing related class 3 suspensions

Example 11-10 shows the first sections for the (group) accounting report that presents the “grand total” of the two jobs that ran concurrently (#OCCURENCES is 2). You can see that the CLASS 2 CPU is 67% of the time the transactions spent in DB2, and CLASS 3 SUSPENSIONS is 24% of the elapsed time spent in DB2 (CLASS 2 ELAPSED TIME).

*Example 11-10 Accounting Report SCOPE(GROUP)*

LOCATION: DB9C GROUP: DB9CG		OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2) ACCOUNTING REPORT - LONG		PAGE: 1-67 REQUESTED FROM: NOT SPECIFIED TO: NOT SPECIFIED INTERVAL FROM: 08/16/09 19:19:58.27 TO: 08/16/09 19:20:03.23	
DB2 VERSION: V9		ORDER: CORRNAME SCOPE: GROUP			
*** GRAND TOTAL ***					
ELAPSED TIME DISTRIBUTION			CLASS 2 TIME DISTRIBUTION		
-----			-----		
APPL	> 1%		CPU	===== 67%	
DB2	===== 75%		SECPU		
SUSP	===== 24%		NOTACC	===== 9%	
			SUSP	===== 24%	
-----					
AVERAGE	APPL(CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	AVERAGE TIME AV.EVENT HIGHLIGHTS
-----					
ELAPSED TIME	1:04.31146	1:03.68805	N/P	LOCK/LATCH(DB2+IRLM)	0.004338 175.00 #OCCURRENCES : 2
....					
CP CPU TIME	43.297039	42.894201	N/P	SER.TASK SWCH	2.648596 856.00 #NORMAL TERMINAT: 2
AGENT	43.297039	42.894201	N/A	UPDATE COMMIT	1.335152 720.50 #DDFRSAF ROLLUP: 0
NONNESTED	1.025071	0.622233	N/P	OPEN/CLOSE	0.957886 46.00 #ABNORMAL TERMIN: 0
STORED PRC	42.188253	42.188253	N/A	SYSLGRNG REC	0.021960 20.00 #CP/X PARALLEL. : 0
UDF	0.000000	0.000000	N/A	EXT/DEL/DEF	0.236223 36.00 #IO PARALLELISM : 0
TRIGGER	0.083715	0.083715	N/A	OTHER SERVICE	0.097374 33.50 #INCREMENT. BIND: 813
PAR.TASKS	0.000000	0.000000	N/A	ARC.LOG(QUIES)	0.000000 0.00 #COMMITTS : 1502
				LOG READ	0.000000 0.00 #ROLLBACKS : 0
SECP CPU	0.000000	N/A	N/A	DRAIN LOCK	0.000000 0.00 #SVPT REQUESTS : 1960
				CLAIM RELEASE	0.000000 0.00 #SVPT RELEASE : 1960
SE CPU TIME	0.000000	0.000000	N/A	PAGE LATCH	0.044487 120.00 #SVPT ROLLBACK : 1960
NONNESTED	0.000000	0.000000	N/A	NOTIFY MSGS	0.000000 0.00 MAX SQL CASC LVL: 2
STORED PROC	0.000000	0.000000	N/A	GLOBAL CONTENTION	10.967994 2872.50 UPDATE/COMMIT : 13.45
UDF	0.000000	0.000000	N/A	COMMIT PH1 WRITE I/O	0.000000 0.00 SYNCH I/O AVG. : 0.000565
TRIGGER	0.000000	0.000000	N/A	ASYNCH CF REQUESTS	0.016275 15.00
				TCP/IP LOB	0.000000 0.00
PAR.TASKS	0.000000	0.000000	N/A	TOTAL CLASS 3	15.194393 6694.50
-----					
SUSPEND TIME	0.000000	15.194393	N/A		

The CLASS 3 SUSPENSIONS section in the accounting report shows the AVERAGE TIME and number of AVEVENTs that the application has been suspended. The locking counters presented in CLASS 3 SUSPENSIONS are:

- ▶ LOCK/LATCH (DB2+IRLM)
- ▶ GLOBAL CONTENTION
- ▶ NOTIFY MSGS
- ▶ PAGE LATCH

### **LOCK/LATCH (DB2+IRLM)**

The LOCK/LATCH field is the average time and number of times that the thread had to wait because a needed resource was held by another task in the *same* DB2 member. It indicates the elapsed time the allied agent has to wait for local locks and latches (DB2 or IRLM). These are the lock and latch suspensions that are also present when you are running in a non-data sharing environment. For more information about this topic, refer to “LOCK/LATCH (DB2+IRLM)” on page 384.

This value does not include global lock suspensions that are waiting for a lock that was held by a transaction on another DB2 member; they appear in the global contention counters.

### **GLOBAL CONTENTION**

The GLOBAL CONTENTION field counts the application delays (wait time and the number of events) when there is a conflict between data sharing members trying to access the same resource at the same time in an compatible manner. In other words, global lock suspensions occur when inter-system communication is required to resolve an IRLM lock or change request.

This value is only provided if accounting class 3 is active. The counter includes both contentions for L-locks and P-locks. In Example 11-10 on page 383, each job spends, V, 10.96 seconds waiting for resources held by the other member. This represents the time the transaction had to wait for the SUSPENDS (IRLM + XES + FALSE) in the data sharing locking section shown in Example 11-11 to be resolved.

*Example 11-11 Accounting report data sharing section*

DATA SHARING	AVERAGE	TOTAL
-----	-----	-----
P/L-LOCKS XES(%)	71.78	N/A
LOCK REQ - PLOCKS	14005.00	28010
UNLOCK REQ - PLOCKS	10429.00	20858
CHANGE REQ - PLOCKS	35.00	70
LOCK REQ - XES	34443.00	68886
UNLOCK REQ - XES	11419.50	22839
CHANGE REQ - XES	889.50	1779
SUSPENDS - IRLM	2512.00	5024
SUSPENDS - XES	22.50	45
SUSPENDS - FALSE	N/A	N/A
INCOMPATIBLE LOCKS	0.00	0
NOTIFY MSGS SENT	29.50	59

More details about the global lock suspensions can be found in the GLOBAL CONTENTION L-LOCKS and GLOBAL CONTENTION P-LOCKS sections of the accounting report, as shown in Example 11-12.

*Example 11-12 Global contention section*

GLOBAL	CONTENTION	L-LOCKS	AVERAGE TIME	AV.EVENT	GLOBAL	CONTENTION	P-LOCKS	AVERAGE TIME	AV.EVENT
L-LOCKS			1.743736	40.50	P-LOCKS			9.224258	2832.00
PARENT	(DB,TS,TAB,PART)		0.000000	0.00	PAGESET/PARTITION			0.086560	32.00
CHILD	(PAGE,ROW)		1.742029	38.00	PAGE			8.806910	2724.50
OTHER			0.001707	2.50	OTHER			0.330787	75.50

### **NOTIFY MSGS**

This is the accumulated elapsed waiting time due to suspensions caused by sending notify messages to other members in the data sharing group. Notify messages are used for a lot of different reasons, so we only provide a few examples.

When a DBD that is cached in the EDM pool (DBDCACHE) on one member changes after executing a DDL statement, it notifies the other DB2 members that the DBD has changed using notify messages. The other members can continue to use their copy of the DBD until they are done with it, but new users will load a fresh copy of the DBD.

Another case where notify messages are used is when the HURBA of a DB2 page set changes. When a data set extend causes the HURBA to increase, the member extending the data set notifies other members to update their HURBA value.

If you experience a high number of notify messages or long notify suspensions, and it is not clear from the application why this is the case, you can trace IFCIDs 44 and 45 (lock suspension and resume) and IFCID 257 (detailed information about the IRLM notify requests) as follows:

```
-START TRACE(P) CLASS(30) IFCID(44,45,257)TDATA(CPU COR DIST) DEST(SMF/GTF)
```

Analysis of such a trace is usually done under the guidance of the IBM support team, as not all the types of messages used by DB2 are externalized.

### **PAGE LATCH**

This is the accumulated waiting time caused by a page latch contention.

For more information, refer to “DB2 latch suspension detailed information” on page 279.

### **Global contention suspensions for L-locks and P-locks**

Continuing the description of data sharing locking information in the accounting report, Example 11-12 shows the details of the reported global contention in the class 3 suspension section. It shows the average time and the average number of times the application was suspended for the different types of L-locks and P-locks.

This global contention section for L-locks and P-locks also exists on the package level, provided accounting trace classes 7 and 8 are active.

### ***L-LOCKS - PARENT***

This is the accumulated global contention wait time for parent L-locks. A parent L-lock can be one of the following types:

- ▶ Database
- ▶ Table space
- ▶ Table
- ▶ Partition

Before locking protocol 2, XES contention (and resolution) on page set/partition L-locks was usually the main contributor to this counter. However, with the advent of locking protocol 2, this type of XES contention has almost been completely eliminated.

### ***L-LOCKS - CHILD***

This is the accumulated global contention wait time for child L-locks. A child L-lock type can be:

- ▶ Page
- ▶ Row

### ***L-LOCKS - OTHER***

This is the accumulated global contention wait time for other types of global L-locks. This is a “none of the above” bucket to catch any type of global L-lock contention that is not the usual parent or child L-lock. The values in this bucket are expected to be small. If that is not the case, you can gather additional information to determine the nature of the other L-lock suspensions using the following TRACE command:

```
-START TRACE(P) CLASS(30) IFCID(44,45)TDATA(CPU COR DIST) DEST(SMF/GTF)
```

This trace gathers information about the lock suspensions and lock resumes in the system. You can limit the amount of trace data being collected by specifying additional filters for the START TRACE command.

One type of lock listed under the OTHER L-lock category is the UTSERIAL lock. This lock is used by utilities to serialize access to SYSUTILX.

### ***P-LOCKS - PAGESET/PARTITION***

This category records the accumulated time for waits due to global contention for page set or partition P-locks. Page set P-locks are always propagated to the coupling facility. Page set/partition P-locks can be negotiated. The negotiation time is also part of the suspension time.

Page set/partition P-lock contention occurs when the inter-DB2 interest level changes, for example, when the page set becomes group buffer pool dependent. When you see high values here, this could mean that either a great deal of work has to be done at that time, propagating locks and registering pages to the structures in the CF, or that access to the CF is slow.

### ***P-LOCKS - PAGE***

This is the accumulated global contention wait time for page P-locks. Page P-locks can be negotiated, and the negotiation time is also part of the suspension time. This category applies to all kinds of page P-locks contention; for data pages (when using row level locking), for space map pages, and on index leaf pages. As page P-lock waits can be an important factor in data sharing processing, there is additional information about them at the group buffer pool level. Refer to “Group buffer pool page P-lock information” on page 388 for more information.

## **P-LOCKS - OTHER**

This is the accumulated global contention wait time for other P-locks. This is a “none of the above” category that captures any P-lock waits that are not page set/partition or page P-locks. The values in this bucket are expected to be small. If that is not the case, you can gather additional information to determine the nature of the other P-lock suspensions by using the following TRACE command:

```
-START TRACE(P) CLASS(30) IFCID(44,45)TDATA(CPU COR DIST) DEST(SMF/GTF)
```

This trace gathers information about the lock suspensions and lock resumes in the system. You can limit the amount of trace data being collected by specifying additional filters on the START TRACE command.

This field includes suspensions for the index tree P-lock (held during index page split for GBP-dependent indexes), castout P-lock, and SKCT/SKPT P-lock.

In summary, the accumulated *global contention* wait time captures all global suspensions for all L-locks and P-Locks. The *global contention L-locks / P-locks* counters have the details. In Example 11-10 on page 383, the average global contention time is 10.967994 seconds. The detailed picture of global suspensions in Example 11-12 on page 385 indicates that the average L-Lock suspension time is 1.74 seconds and is due to the suspension on child locks (PAGE or ROW), and that the vast majority of the global contention is for P-locks at 9.22 seconds, and this is almost completely because of page P-locks contention.

## **Data sharing locking activity section**

Before we look at the group buffer pool information (to take a closer look at the type of page P-locks involved in the global contentions) in Example 11-12 on page 385, we first have a look at the data sharing locking activity section of the accounting report.

The data sharing locking activity section reported in the accounting report is shown in Example 11-11 on page 384. These fields are identical to the ones in the OMEGAMON PE statistics reports (refer to “Data sharing locking section” on page 375).

During our tests, the DB2 systems were dedicated to our scenario jobs, so the accounting information matches the statistics report. For example, in the statistics reports, the total SUSPENDS - IRLM GLOBAL CONT was 5044.0 system-wide (refer to Example 11-7 on page 375). In the data sharing activity section of the accounting report (Example 11-11 on page 384), the total value for both workload jobs is 5024.

Note that the XES lock/unlock/change requests can be underestimated if a lot of heuristic conversions occur. This is the same problem that we described for the data sharing locking counters in the statistics report. Refer to “Data sharing locking section” on page 375 for more details.

Note that the SUSPENDS - FALSE counter shows N/A. At the time of writing, DB2 cannot distinguish between false contentions and lock requests that were converted from sync to async by the z/OS heuristic algorithm. To avoid reporting confusing values, OMEGAMON PE stopped reporting this information. This should change when the PTFs for APARs PK85159 and PK85543, still open at the time of writing, become available.

## Locking activity section

The accounting report using SCOPE(GROUP) shows the L-Lock locking activity (Example 11-13), which is the type of information we also have for non-data sharing systems. For more information about these counters, refer to 8.3, “Periodic monitoring for concurrency problems” on page 227.

*Example 11-13 Accounting report locking section*

LOCKING	AVERAGE	TOTAL
-----	-----	-----
TIMEOUTS	0.00	0
DEADLOCKS	0.00	0
ESCAL.(SHARED)	0.00	0
ESCAL.(EXCLUS)	0.00	0
MAX PG/ROW LOCKS HELD	35.50	36
LOCK REQUEST	33977.00	67954
UNLOCK REQUEST	4439.00	8878
QUERY REQUEST	0.50	1
CHANGE REQUEST	2399.50	4799
OTHER REQUEST	0.00	0
TOTAL SUSPENSIONS	17.00	34
LOCK SUSPENSIONS	0.00	0
IRLM LATCH SUSPENS.	8.00	16
OTHER SUSPENS.	9.00	18

## Group buffer pool page P-lock information

The GLOB CONT PAGE P-LOCK subsection in Example 11-12 on page 385 shows that most of global suspension time is for page P-locks. The group buffer pool activity section of the accounting report provides more details about the types of page P-locks that are used (Example 11-14). In this case, the page P-Locks suspensions is for SPACE MAP PAGES and INDEX LEAF PAGES.

*Example 11-14 Group buffer pool section*

GROUP TOT4K	AVERAGE	TOTAL
-----	-----	-----
...		
PG P-LOCK LOCK REQ	13033.00	26066
SPACE MAP PAGES	3231.50	6463
DATA PAGES	50.50	101
INDEX LEAF PAGES	9751.00	19502
PG P-LOCK UNLOCK REQ	10373.00	20746
PG P-LOCK LOCK SUSP	2602.50	5205
<b>SPACE MAP PAGES</b>	<b>1700.00</b>	<b>3400</b>
DATA PAGES	1.00	2
<b>INDEX LEAF PAGES</b>	<b>901.50</b>	<b>1803</b>
...		

The GBP section contains much information about the type of activity that occurs for each GBP. As this publication focuses on locking and serialization, we only discuss the counters related to locking, more specifically those related to page P-lock activity.

### ***PG P-LOCK LOCK REQ***

This is the number of page P-lock lock requests for this group buffer pool (in our case, the total of all 4K group buffer pools). This counter includes subsections for the type of the requested page P-lock:

- ▶ **SPACE MAP**

The space map page P-lock request is used to serialize updates between DB2 members on the same space map page, for example, when INSERTing data in same table by multiple members and that data free space in the same space map has to be updated.

- ▶ **DATA PAGES**

A data page page P-lock request happens when there is row-level locking.

- ▶ **INDEX LEAF PAGES**

Index leaf page P-lock requests are used when more than one member of a DB2 data sharing group is inserting or updating data on the same table data and indexes must be updated with the new data.

### ***PG P-LOCK UNLOCK REQ***

This is the number of page P-lock unlock requests.

### ***PG P-LOCK LOCK SUSP***

This is the sum of all page P-lock lock suspensions. This section also has breakdown sections for the space map, data pages, and leaf pages suspensions that we discussed at the beginning of this section.

## **11.4 Monitoring data sharing locks using RMF reports**

The RMF coupling facility activity report describes the activity for all structures in the coupling facility for a given time period. A CF activity report can be generated using the RMF post processor (based on the RMF SMF type 74 subtype 4 records) using the following input command:

```
//SYSIN DD *  
        SYSRPTS(CF)  
        SYSOUT(H)  
/*
```

The CF activity report provides information about each coupling facility in the sysplex. The reports consists of a number of different sections:

- ▶ The Coupling Facility usage summary, which contains a high level overview of all the structures that are allocated in a particular CF.
- ▶ The Coupling Facility structure activity. This section reports on the activity of an individual CF structure.
- ▶ The Subchannel activity, which provides information about subchannels that attach the CF to the different LPARs that host the users of the structure in that CF.
- ▶ The CF to CF activity, which is only present when system-managed duplexing is in effect.

Example 11-15 shows the *structure activity* section for the DB9CG\_LOCK1 lock structure of our data sharing group. The information is for a 10 minute interval starting at 12:19.38. The report lists all systems (that are present in the input file) that have sent requests to this structure during the reporting interval, in this case, SC63 and SC64.

Example 11-15 RMF coupling facility activity report

COUPLING FACILITY ACTIVITY													
z/OS V1R11		SYSPLEX SANDBOX				DATE 08/11/2009				INTERVAL 010.00.000			
		RPT VERSION V1R11 RMF				TIME 12.19.38				CYCLE 01.000 SECONDS			
-----													
COUPLING FACILITY NAME = CF1													
-----													
STRUCTURE NAME = DB9CG_LOCK1      TYPE = LOCK      STATUS = ACTIVE													
-----													
# REQ		REQUESTS		SERV TIME(MIC)		REASON		DELATED REQUESTS		AVG TIME(MIC)		EXTERNAL REQUEST	
SYSTEM	TOTAL	#	% OF	-SERV	TIME(MIC)			#	% OF	----	TIME(MIC)	----	CONTENTIONS
NAME	AVG/SEC	REQ	ALL	AVG	STD_DEV			REQ	REQ	/DEL	STD_DEV	/ALL	
SC63	164K[Z]	SYNC	164K[Y]	100	5.0[W]	6.7	NO SCH	0	0.0	0.0	0.0	0.0	REQ TOTAL      213K[A]
	273.7	ASYN	0 [X]	0.0	0.0	0.0	PR WT	0	0.0	0.0	0.0	0.0	REQ DEFERRED      2725
		CHNGD	0	0.0	INCLUDED IN ASYN		PR CMP	0	0.0	0.0	0.0	0.0	-CONT      2725 [B]
													-FALSE CONT      772 [C]
SC64	129K	SYNC	129K	100	5.1	8.0	NO SCH	0	0.0	0.0	0.0	0.0	REQ TOTAL      162K
	215.0	ASYN	0	0.0	0.0	0.0	PR WT	0	0.0	0.0	0.0	0.0	REQ DEFERRED      2794
		CHNGD	0	0.0	INCLUDED IN ASYN		PR CMP	0	0.0	0.0	0.0	0.0	-CONT      2792
													-FALSE CONT      526
-----													
TOTAL	129K	SYNC	293K	100	5.05	7.35	NO SCH	0	0.0	0.0	0.0	0.0	REQ TOTAL      375K
	488.7	ASYN	0	0.0	0.0	0.0	PR WT	0	0.0	0.0	0.0	0.0	REQ DEFERRED      5519
		CHNGD	0	0.0			PR CMP	0	0.0	0.0	0.0	0.0	-CONT      5585
													-FALSE CONT      1298

The # REQ TOTAL (total number of requests) from system SC63 in this interval is 164K [Z], or 372.7 per second. In this case, all requests were processed synchronously (SYNC 164K [Y] and ASYN 0 [X]). Remember that DB2 or IRLM may ask for the IXLLOCK request to be processed synchronously, but the z/OS heuristic algorithm can convert the request to async. If that happens frequently, the number of ASYN requests [X] will be high. The average service time for the lock requests in this example are 5.0  $\mu$ sec [W], which is a very good average response time.

**Tip:** With current technology, synchronous CF requests are expected to take less than 10  $\mu$ sec, usually in the 8-10  $\mu$ sec range. Asynchronous requests, on the other hand, are expected to be in the 100  $\mu$ sec range, and usually also show a wider degree of variation (indicated by a high STD\_DEV).

When requests are delayed, there will be non-zero values in the DELAYED REQUESTS section. The most common reason is no subchannel (NO SCH). When there are many subchannel busy conditions, you can also expect a high number of CHNGD requests. This is usually an indication that there are too many requests for the number of available channels. When system-managed duplexing is used, it is normal to see delayed requests with a reason subchannel wait time (PR WT) and waiting-for-peer-completion time (PR CMP).

The REQ TOTAL [A] on the right hand side are the number of external requests, which is the the number of lock requests that IRLM sent to XES.



**Tip:** The number reported on the right side of the report, in the REQ TOTAL [A] field under EXTERNAL REQUEST, is the number of lock requests (lock, change, and unlock requests) sent to XES by IRLM. It includes the synchronous and asynchronous requests.

The number on the left side of the report, under # REQ TOTAL [Z], is the number of those requests that were *actually* sent to the Coupling Facility. The difference between the two can be quite large. This difference is accounted by:

- ▶ Batch unlock requests: When DB2 wants to release many locks at one time, it sends a single IXLLOCK request to XES, providing a list of locks to be released. Each lock to be released increments the External Request count by one, but many locks can be released by a single request to the Coupling Facility. So, a large amount of batch unlock activity (typical of a batch environment) can result in a larger count on the right side of the report than on the left
- ▶ Contention: If a lock entry is already known to be in contention when a lock request is received by XES, the request will be passed to the global lock manager for that lock entry, rather than being passed to the Coupling Facility. So, as the level of lock contention increases, you would expect to see a growing difference between the number of lock requests sent to XES (on the right side of the report) and the number sent to the lock structure (on the left side).

There were 2727 [B] requests that were deferred because of contention. Of those, 772 [C] turned out to be false contention. Note that RMF cannot distinguish between XES contention and real or IRLM contention, so from a CF point of view, the number of “real” contentions is [B]-[C].

You can calculate the amount of global and false contention based on the CF activity report as follows:

- ▶ [A]: Total number of lock-related requests.
- ▶ [B]: Number of requests that were deferred because of contention.
- ▶ [C]: The number of deferred requests that were caused by false contention.
- ▶ Global contention =  $[B] / [A] * 100$ .
- ▶ False contention =  $[C] / [A] * 100$ .

In Example 11-15 on page 390, the DB2 member on system SC63 has a global contention rate value of 1,25% ( $2725 / 213K * 100$ ). The false contention rate is 0,35% ( $772 / 213K * 100$ ).

**Note:** As a general ROT, keep global contention below 3 to 5%, and false contention to less than 50% of the global contention.

Locking protocol 2 greatly reduces the amount of XES lock contention, which means that there is also likely to be a reduction in false contention, resulting in a reduced amount of global lock contention.

The bottom of the RMF “coupling facility usage summary” section provides information about the coupling facility utilization rate. You should check the AVERAGE CF UTILIZATION (% BUSY) field.

**Tip:** The coupling facility processor(s) should be able to handle the workload with availability and without having a very high utilization rate. Start with two engines and add another processor when CPU utilization reaches 50%.

## 11.5 Data sharing global contention scenarios

In this section, we describe three scenarios where the main purpose is to illustrate and point out how to discover and avoid certain types of global lock contention and data sharing global locking in general.

The workload was generated using the IBM Workload Generator Version 1.5 tool. An identical workload was executed on both members of the data sharing group, intentionally creating contention problems. The purpose of the workload is not to show the general performance of our DB2 system or how the workload behaves in a data sharing environment, but rather show the differences in locking attributes and the associated processing impact.

### 11.5.1 Global lock avoidance

One of the most important considerations related to global locking in a DB2 data sharing environment is to take advantage of lock avoidance. Lock avoidance can be achieved by using the ISOLATION(CS) CURRENTDATA(NO) BIND options (this is the default in DB2 9). Watch out for long running unit of works, as they can reduce the effectiveness of lock avoidance. Refer to 10.12, “Lock avoidance in data sharing” on page 362 for details.

This workload scenario compares using BIND ISOLATION(CS) with CURRENTDATA(YES) versus CURRENTDATA(NO) to demonstrate the effect of effective lock avoidance.

The workload is a stored procedure workload that performs concurrent database access from both DB2 members using the following, several SQL statements:

```
BIND PACKAGE(GLWSAMP) MEM(EMPANO )
QUAL(GLWSAMP) LIB('DB2CFG.GLW.SGLWDBRM')
SQLERROR(N) VALIDATE(R) ISOLATION(CS) EXPLAIN(Y) ACTION(REPLACE)
DEGREE(1) KEEPYNAMIC(N) NOREOPT(VARS) CURRENTDATA(NO);
```

The stored procedures packages were bound with CURRENTDATA YES or NO respectively for each test run

The OMEGAMON PE accounting report was created using the following input commands:

```
ACCOUNTING
        REPORT
        SCOPE(GROUP)
        ORDER(CORRNAME)
        LAYOUT (LONG)
                DDNAME(ACRPTDD)
EXEC
```

Example 11-16 shows a statement mix for our workload environment using the accounting report's SQL DML section.

*Example 11-16 SQL DML accounting report section: example 1*

---

SQL DML	AVERAGE	TOTAL
-----	-----	-----
SELECT	18965.00	37930
INSERT	13075.00	26150
UPDATE	1761.00	3522
MERGE	0.00	0
DELETE	51.00	102

DESCRIBE	40.00	80
DESC.TBL	0.00	0
PREPARE	240.00	480
OPEN	2857.50	5715
FETCH	3244.4K	6488854
CLOSE	2832.50	5665

DML-ALL	3284.2K	6568498
---------	---------	---------

Note that our workload includes a fairly high number of INSERT, UPDATE, and DELETE SQL statements. These DML statements are not subjects of lock avoidance, but our workload show improvements when activating lock avoidance.

In Table 11-2, you can see a substantial difference in the amount of logical locking and data sharing locking requests when the lock avoidance is in effect. The L-lock lock and unlock requests are much higher when using CURRENTDATA(YES). The number of L-locks and P-locks requested by XES is also much higher (LOCK REQ - XES) when using CD(Y), but note that in this workload the P-locks requests are almost the same number in both cases, so the increase is just because of the additional L-locks that need to be propagated. The number of suspensions in the data sharing locking section is 8391.00 compared with 6891.00 when using CURRENTDATA(NO). Thus, the fewer locks there are, the fewer chances there are to get suspended.

Table 11-2 Comparison for CD=NO and CD=YES with ISOLATION(CS)

Item	ISO(CS) and CD(YES)	ISO(CS) and CD(NO)
CL2 CPU TIME	1:35.88153	1:31.44536
DML- ALL	3284.2K	3284.2K
L-LOCK REQUEST	255.5K	92.4K
L-LOCK UNLOCK REQUEST	179.6K	17.8K
TOTAL SUSPENSIONS (IRLM LOCK, IRLM LATCH, OTHER)	84.50	58.50
P/L-LOCKS XES(%)	88.75%	75.5%
LOCK REQ - PLOCKS	40,7K	40K
UNLOCK REQ- PLOCKS	28,8K	29.3K
LOCK REQ - XES	262.9K	100K
UNLOCK REQ - XES	194.9K	34.8K
SUSPENDS - IRLM	8391.00	6891.00
SUSPENDS - XES	20.00	19.00
SUSPENDS - FALSE	N/A	N/A

Also note that there are almost no XES contentions (as we are running with locking protocol 2).

**Tip:** Lock avoidance may not be working effectively if the number of unlock requests / commit is high (greater than 5 per commit) and the number of unlock request is greater than one-third of the number of lock requests. This ROT applies to both non-data sharing and data sharing environments.

## 11.5.2 LOCKSIZE ROW versus LOCKSIZE PAGE

The intention of this workload scenario is to compare row level locking against page level locking in a DB2 data sharing environment.

Because of the potential increase in page P-lock activity when using row level locking in a data sharing environment, you should always carefully evaluate the use of LOCKSIZE ROW table spaces before using it in a DB2 data sharing environment, as the amount of page P-lock activity is likely to increase the data sharing processing impact.

Some applications have fairly small tables that demand a very high degree of concurrency. To avoid locking issues, these tables often use LOCKSIZE ROW.

When running this type of application in a data sharing environment, we highly recommend evaluating the impact of using the row-level locking and consider an alternative design using LOCKSIZE PAGE and MAXROWS 1 to achieve the row-level locking concurrency (as there is only one row per page), but without the increased data sharing impact of using page P-locks.

A group buffer pool dependent table space defined with row-level locking uses page P-lock on the data pages as a global latch mechanism to control the physical consistency of the page. Refer to 10.7.1, “Page P-locks usage by row level locking” on page 330 for details about row level locking behavior in data sharing.

The scenario we used here is using a workload with mainly update activity against objects that use row level locking. The main purpose of the scenario is to help evaluate the possibility of using MAXROWS 1 with LOCKSIZE PAGE as an alternative to using LOCKSIZE ROW. The scenario has the following characteristics:

- ▶ The same amount of SQL UPDATE statements are executed on both DB2 members D9C1 and D9C2.
- ▶ Table GLWTEMP is small and only has 100 rows.
- ▶ The scenario also includes SQL SELECT statements.
- ▶ Table spaces are defined with LOCKSIZE ROW versus LOCKSIZE PAGE.
- ▶ The table space uses buffer pool (BP2) and group buffer pool (GBP2).
- ▶ The index space uses buffer pool (BP3) and group buffer pool (GBP3).

The scenario uses a table with 100 rows. The MAXROWS value of one on the table spaces simulates the row locking and can be used as a solution for small tables. You can get the benefits of row locking without the data page P-lock contention. Note that a new MAXROWS value does not take effect until you run REORG on the table space.

This scenario uses the stored procedures workload to perform concurrent SQL UPDATE statements on both DB2 data sharing members against the GLWTEMP table. The workload calls the EMPUPR stored procedure that executes the updates statements 999 times.

Both runs execute the same SQL statement on both members. The only difference between the two runs is the LOCKSIZE value of ROW versus PAGE MAXROWS 1. Example 11-17 shows the SQL statements executed for both runs of this scenario and was extracted from the SQL DML section of the ACCOUNTING REPORT.

*Example 11-17 SQL DML accounting report section: example 2*

SQL DML	AVERAGE	TOTAL
-----	-----	-----
SELECT	5994.00	23976
INSERT	0.00	0
UPDATE	999.00	3996
MERGE	0.00	0
DELETE	1.00	4
DESCRIBE	40.00	160
DESC.TBL	0.00	0
PREPARE	40.00	160
OPEN	40.00	160
FETCH	82.00	328
CLOSE	40.00	160
DML-ALL	7236.00	28944

The workload generator performed 3996 random updates in total against the 100 row GLWTEMP table across both members of the data sharing group. Table 11-3 shows the relevant differences in the accounting counters when comparing LOCKSIZE ROW and PAGE plus MAXROWS equal to one locking scenario in data sharing. The values we present are the accounting report AVERAGE counters.

Table 11-3 shows that the L-LOCK REQUESTs are almost the same (identical workload), but the P-LOCK REQUESTs are much higher for the LOCKSIZE ROW case. There is also an increase in the number of requests that have been sent synchronously to XES, as reported by the LOCK REQ - XES counter.

*Table 11-3 Comparison of LOCKSIZE ROW versus LOCKSIZE PAGE MAXROWS 1*

Item	LOCKSIZE ROW	LOCKSIZE PAGE MAXROWS 1
CL2 CPU TIME	1.055689	1.112246
DML-ALL	7236.00	7236.00
CL3 GLOBAL CONTENTION	0.202883	0.016607
L-LOCK REQUEST	12134.25	12145.25
TOTAL SUSPENSIONS (IRLM LOCK, IRLM LATCH, OTHER)	1.25	2.50
P/L-LOCKS XES(%)	59.23	56.71
LOCK REQ - PLOCKS	817.00	42.00
LOCK REQ - XES	7670.50	6926.50
SUSPENDS - IRLM	76.75	4.25
SUSPENDS - XES	1.00	1.00

Item	LOCKSIZE ROW	LOCKSIZE PAGE MAXROWS 1
SUSPENDS - FALSE	N/A	N/A

In the accounting CL3 SUSPEND TIME section, the CL3-GLOBAL CONTENTION time for the LOCKSIZE PAGE MAXROWS1 scenario is only 8.2% of the CL3-GLOBAL CONTENTION time for the LOCKSIZE ROW in this scenarios (16 milliseconds for the page-level workload versus 202 milliseconds for the row-level workload).

To provide a better insight in the class 3 global contention time, Example 11-8 on page 379 shows the global contention section for both scenarios. The global contention for L-locks when using row-level locking is 6 milliseconds, all of which reported for row L-locks. The row-level locking workload also shows 173 milliseconds of page P-lock global contention. The page-level locking scenario with MAXROWS=1 shows no significant page P-lock contention. Note that the very small value for page P-lock is because the workload uses sequences to generate the next transaction value. It is not related to row or page level locking and it is shown in GROUP BP0 as the SYSIBM.SYSSEQUENCE catalog table space uses that local and group bufferpool.

*Example 11-18 Global contention section: example 1*

**LOCKSIZE ROW**

GLOBAL	CONTENTION	L-LOCKS	AVERAGE TIME	AV.EVENT
-----				
L-LOCKS			0.006168	8.00
PARENT	(DB,TS,TAB,PART)		0.000000	0.00
CHILD	(PAGE,ROW)		0.006168	8.00
OTHER			0.000000	0.00
GLOBAL	CONTENTION	P-LOCKS	AVERAGE TIME	AV.EVENT
-----				
P-LOCKS			0.196716	110.00
PAGESET/PARTITION			0.013450	1.00
<b>PAGE</b>			<b>0.173472</b>	<b>104.50</b>
OTHER			0.009794	4.50

**LOCKSIZE PAGE plus MAXROWS=1**

GLOBAL	CONTENTION	L-LOCKS	AVERAGE TIME	AV.EVENT
-----				
L-LOCKS			0.001709	1.50
PARENT	(DB,TS,TAB,PART)		0.001709	1.50
CHILD	(PAGE,ROW)		0.000000	0.00
OTHER			0.000000	0.00
GLOBAL	CONTENTION	P-LOCKS	AVERAGE TIME	AV.EVENT
-----				
P-LOCKS			0.014898	5.75
PAGESET/PARTITION			0.002421	1.00
PAGE			0.000768	0.25
OTHER			0.011710	4.50

Example 11-19 shows the page P-lock counters of the group buffer pool section for this scenario. The page P-Lock lock requests on data pages in table space group buffer pool (GBP2) for the row-level locking scenario is 749.25 on average, and the page P-lock suspensions for data pages is 104.00 msec.. Note that the AV.EVENT for PAGE in the global contention P-lock section above is 104.50 and has 173.00 msec. of suspension time. The page-level workload does not have page P-lock requests, as expected.

*Example 11-19 Group buffer pool section: example 1*

---

**LOCKSIZE ROW**

GROUP BP2	AVERAGE	TOTAL
-----	-----	-----
...		
PG P-LOCK LOCK REQ	749.25	2997
SPACE MAP PAGES	0.00	0
<b>DATA PAGES</b>	<b>749.25</b>	<b>2997</b>
INDEX LEAF PAGES	0.00	0
PG P-LOCK UNLOCK REQ	703.75	2815
PG P-LOCK LOCK SUSP	104.00	416
SPACE MAP PAGES	0.00	0
<b>DATA PAGES</b>	<b>104.00</b>	<b>416</b>
INDEX LEAF PAGES	0.00	0
...		

**LOCKSIZE PAGE plus MAXROWS=1**

GROUP BP2	AVERAGE	TOTAL
-----	-----	-----
...		
PG P-LOCK LOCK REQ	0.00	
SPACE MAP PAGES	0.00	
DATA PAGES	0.00	
INDEX LEAF PAGES	0.00	
PG P-LOCK UNLOCK REQ	0.00	
PG P-LOCK LOCK SUSP	0.00	
SPACE MAP PAGES	0.00	
DATA PAGES	0.00	
INDEX LEAF PAGES	0.00	
...		

---

The usage of LOCKSIZE ROW should be monitored for resource processing impact (including CPU) in the DB2 data sharing environments.

However, it should be noted that using LOCKSIZE PAGE MAXROWS 1 is not a panacea for all LOCKSIZE ROW scenarios that experience page P-lock contention and an increased resource usage. It should be carefully evaluated, but it can be a good alternative for relatively small tables so that the additional space usage or decreased buffer pool efficiency does not impact the application's or even the system's overall performance.

### 11.5.3 High INSERT data scenario

The workloads used in this scenario highlight some of the performance effects of a high number of parallel inserts in a DB2 data sharing environment.

As before, the scenario uses the workload to perform 10,000 INSERTS statements into the GLWTEMP table from each DB2 data sharing member (jobs DB2INS63 and DB2INS64). The jobs insert rows sequentially by EMP\_NO, with an ever increasing EMP\_NO. The workload calls the EMPADD stored procedure 10,000 times.

The workload was run three times with different DDL options.

Example 11-20 shows an excerpt of the workload job output from one of the DB2 members.

#### *Example 11-20 Stored procedures workload output*

---

```
GLWR142I: Stored Procedure call summary
.....
DPTBAL    did not run
DPTDEL    did not run
DPTLCK    did not run
DPTMGR    did not run
DPTUPD    did not run
DPTUPR    did not run
EMPADD    ran 10000 times at an average elapsed of    0.006 seconds
EMPDEL    did not run
.....
Total calls= 10000 ; Runtime=      1.0 Minutes
Transaction rate=  173.9 trans/sec
READY
END
```

---

As before, we use the DB2 accounting report SCOPE(GROUP) report. This way, you can analyze the run at the member level, but also at the group (refer to Example 11-21).

#### *Example 11-21 Accounting report with SCOPE(GROUP) example*

---

```
1  LOCATION: DB9C          OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)          PAGE: 1-1
   GROUP: DB9CG          ACCOUNTING REPORT - LONG          REQUESTED FROM: NOT SPECIFIED
                                     TO: NOT SPECIFIED
                                     INTERVAL FROM: 08/16/09 19:19:58.27
DB2 VERSION: V9          ORDER: CORRNAME          TO: 08/16/09 19:20:03.23
                           SCOPE: GROUP

CORRNAME: DB2INS63  MEMBER: D9C1

ELAPSED TIME DISTRIBUTION          CLASS 2 TIME DISTRIBUTION
-----
APPL  |> 1%
DB2   |===== 76%
SUSP  |===== 23%

.....
.....
.....
ORDER: CORRNAME          INTERVAL FROM: 08/16/09 19:19:58.27
DB2 VERSION: V9          SCOPE: GROUP          TO: 08/16/09 19:20:03.23

*** GRAND TOTAL ***
ELAPSED TIME DISTRIBUTION          CLASS 2 TIME DISTRIBUTION
-----
APPL  |> 1%
DB2   |===== 75%
SUSP  |===== 24%

.....
.....
.....
CPU   |===== 69%
SECPU |===== 8%
NOTACC |===== 23%
SUSP  |===== 23%
```

---



Example 11-22 shows the SQL statements that were executed by the insert workload run (runmode(fixed)). You can see that the TOTAL is two times the AVERAGE, as the #OCCURRENCES is two and both jobs run the exact same (number of) SQL statements.

*Example 11-22 SQL DML accounting report section: example 3*

SQL DML	AVERAGE	TOTAL
-----	-----	-----
SELECT	130.0K	260000
INSERT	10000.00	20000
UPDATE	0.00	0
MERGE	0.00	0
DELETE	1.00	2
DESCRIBE	40.50	81
DESC.TBL	0.00	0
PREPARE	40.50	81
OPEN	40.50	81
FETCH	82.50	165
CLOSE	40.50	81
DML-ALL	140.2K	280491

The three scenarios use the same table space, GLWSTEMP with Numparts 4, and table GLWTTEMP with three indexes. The inserts are performed by EMP\_NO, and the table space and index options that are used are listed in Table 11-4.

*Table 11-4 High INSERT scenario options*

Scenario #1	Scenario #2	Scenario #3
NO MEMBER CLUSTER TRACKMOD YES	NO MEMBER CLUSTER TRACKMOD YES	MEMBER CLUSTER TRACKMOD NO
INDEX with index columns ASC	INDEX with column EMP_NO RANDOM	INDEX with column EMP_NO RANDOM

Table 11-5 compares the results using the group level sections (grand total) of the accounting SCOPE(GROUP) report.

*Table 11-5 High INSERT scenarios comparison*

Item	SCENARIO #1	SCENARIO #2	SCENARIO #3
CL2 CPU TIME	27.42	27.54	27.61
CL3 GLOBAL CONTENTION	11.71	4.01	0.57
#INSERTS	10000	10000	10000
L-LOCK REQUEST	192.6K	194.8K	192.0K
TOTAL SUSPENSIONS (IRLM LOCK, IRLM LATCH, OTHER)	35.00	27.50	33.00
P/L-LOCKS XES(%)	66.30%	66.68%	66.21%
LOCK REQ - P LOCKS	30753.50	31255.50	30538.00
LOCK REQ - XES	148.1K	150.7K	147.3K
SUSPENDS - IRLM	1748.00	231.50	55.50
SUSPENDS - XES	2.50	2.50	2.50
SUSPENDS - FALSE	N/A	N/A	N/A

At a high level, all three scenarios in Table 11-5 have a similar number of (L-)lock requests, P-lock requests, and the number of lock request sent to XES. The percentage for the P-lock and L-lock requests propagated to the coupling facility is around 66% and the number of suspensions in the (L-) locking activity section also have almost the same value (35, 27.5, and 33). However, the class 3 global contention suspension time was 11.7 seconds in the first scenario (without MEMBER CLUSTER, and before using the index RANDOM option). In scenario #2, the global contention time was reduced to 4 seconds (after implementing the RANDOM index option). In scenario #3, the class 3 global contention suspension dropped further to 0.57 seconds (after USING MEMBER CLUSTER and TRACKMOD NO). Note that the IRLM suspends (or real contention) has also decreased, and that the number XES contention suspension is only 2.5. Locking protocol 2 can significantly reduce XES (and false) contention.

### Scenario #1

Let us have a closer look at the global contention section for scenario #1 to better understand what type of global contention is really responsible for the 11.71 seconds of global contention in scenario #1. Example 11-23 on page 401 shows that out of the 11.71 seconds of class 3 global contention suspension time, 11.6689 seconds are from the P-locks, and the L-locks global contention is only 44.94 milliseconds. Within P-locks global contention, the vast majority is for page P-locks, which account for 10.028188 seconds of suspension time, during 2829 suspension events (on average).

Example 11-23 Global contention section: example

GLOBAL	CONTENTION	L-LOCKS	AVERAGE TIME	AV.EVENT
-----			-----	-----
L-LOCKS			0.044943	5.50
PARENT	(DB,TS,TAB,PART)		0.003001	3.00
CHILD	(PAGE,ROW)		0.001325	1.50
OTHER			0.040617	1.00
GLOBAL	CONTENTION	P-LOCKS	AVERAGE TIME	AV.EVENT
-----			-----	-----
P-LOCKS			11.668905	2860.00
PAGESET/PARTITION			0.043955	14.00
<b>PAGE</b>			<b>10.028188</b>	<b>2829.00</b>
OTHER			1.596763	17.00

The page P-locks counters from the group buffer pool activity section are shown in Example 11-24. Group buffer pool BP2 is used for table spaces and accounts for 398 page P-lock suspensions for space map pages. Group buffer pool BP3 show 2434 page P-lock suspensions (on average), all for index leaf pages.

Example 11-24 Group buffer pool P-lock activity

GROUP BP2	AVERAGE	TOTAL	GROUP BP3	AVERAGE
-----	-----	-----	-----	-----
...			...	
PG P-LOCK LOCK REQ	1289.50	2579	PG P-LOCK LOCK REQ	28869.50
SPACE MAP PAGES	1289.50	2579	SPACE MAP PAGES	0.00
DATA PAGES	0.00	0	DATA PAGES	0.00
INDEX LEAF PAGES	0.00	0	INDEX LEAF PAGES	28869.50
PG P-LOCK UNLOCK REQ	1135.00	2270	PG P-LOCK UNLOCK REQ	28094.50
PG P-LOCK LOCK SUSP	398.00	796	PG P-LOCK LOCK SUSP	2434.00
SPACE MAP PAGES	398.00	796	SPACE MAP PAGES	0.00
DATA PAGES	0.00	0	DATA PAGES	0.00
INDEX LEAF PAGES	0.00	0	INDEX LEAF PAGES	2434.00
...			...	

## Scenario #2

In scenario #2, we tried to reduce the index leaf page P-lock contention by using the RANDOM keyword in the index's EMP\_NO column. RANDOM was introduced in DB2 9 for z/OS. You can define an index column as ASC, DESC, or RANDOM. When using RANDOM, the index column value is randomized and put in a random order by the column. This way, instead of inserting ever increasing EMP\_NO values that all have to be inserted into the same index leaf page, the values that are to be inserted are random and go into different pages (provided the randomizer does a good job), thereby minimizing the index leaf page hot spot.

Example 11-25 illustrates the results of scenario #2. The data shows a big reduction in index leaf pages P-lock suspension for our insert workload.

*Example 11-25 Global contention and group buffer pool page P-locks*

GLOBAL	CONTENTION	P-LOCKS	AVERAGE TIME	AV.EVENT
-----				
P-LOCKS			3.969328	422.00
PAGESET/PARTITION			0.026166	4.00
PAGE			3.892778	406.00
OTHER			0.050384	12.00
GROUP BP3		AVERAGE	TOTAL	
-----				
...				
PG P-LOCK LOCK REQ		29535.00	59070	
SPACE MAP PAGES		0.00	0	
DATA PAGES		0.00	0	
INDEX LEAF PAGES		29535.00	59070	
PG P-LOCK UNLOCK REQ		29592.00	59184	
<b>PG P-LOCK LOCK SUSP</b>		<b>56.00</b>	<b>112</b>	
SPACE MAP PAGES		0.00	0	
DATA PAGES		0.00	0	
<b>INDEX LEAF PAGES</b>		<b>56.00</b>	<b>112</b>	
WRITE AND REGISTER		29486.00	58972	

In our case, using a random index allowed us to significantly reduce the number of index leaf page P-lock suspensions. However, using a random index is most certainly not the answer to every index leaf page P-lock problem. Random indexes have to be used with great care. They have the ability to significantly reduce page P-lock contention, for example, during massive inserts, but they also have the potential to create problems for other applications that need to process the data afterwards. Random indexes do not support range predicates and can result in a decrease in buffer pool hit ratio, as index keys with similar key value will now be spread out all over the index.

In our database, however, using a random index is acceptable. The EMP\_NO column is only used to look up an employee when we know the number, that is, a single row retrieval. This is perfectly okay for a random index. The EMP\_NO is a “meaningless” number and is not used for any type of random or sequential processing.

A case where a random index could have a negative impact is when you insert data based on an index with a time stamp. In this case, the index entries are always ascending and can cause index leaf page contention. Randomizing this key column can help reduce this problem during insert. However, when at the end of the day you have to process all the data for the day (based on this insert time stamp column that is now random), all index keys for the current day will be all over the index and can seriously impact this type of sequential processing. Therefore, use random indexes with care.

### Scenario #3

Scenario #3 uses the table space MEMBER CLUSTER option to reduce the space map page P-locks suspensions (we also use the TRACKMOD NO option). Example 11-26 on page 403 shows the global P-lock contention and the group buffer pool activity for BP2 for our scenario high INSERT scenario after changing to MEMBER CLUSTER.

*Example 11-26 Global contention and group buffer pool P-lock activity*

GLOBAL	CONTENTION	P-LOCKS	AVERAGE TIME	AV.EVENT
-----			-----	-----
P-LOCKS			0.506064	80.00
PAGESET/PARTITION			0.041099	14.00
PAGE			0.415626	56.00
OTHER			0.049340	10.00
GROUP BP2		AVERAGE		
-----			-----	
....				
PG P-LOCK LOCK REQ		839.00		
SPACE MAP PAGES		837.50		
DATA PAGES		1.50		
INDEX LEAF PAGES		0.00		
PG P-LOCK UNLOCK REQ		561.50		
<b>PG P-LOCK LOCK SUSP</b>		<b>2.00</b>		
<b>SPACE MAP PAGES</b>		<b>2.00</b>		
DATA PAGES		0.00		
INDEX LEAF PAGES		0.00		
WRITE AND REGISTER		8958.00		
....				

Note that the number of space map contentions changed from 398 (in scenario #1) to 2, and that the global page P-lock contention time was further reduced from 3.89 (in scenario #2) to 0.41 after changing to MEMBER CLUSTER and TRACKMOD NO.

Again, as is the case for random indexes, MEMBER CLUSTER is not the answer to every problem. It will help insert performance, but may have some impact on processes that have to retrieve the inserted data afterwards.

For a more detailed discussion about the MEMBER CLUSTER design alternative, refer to “MEMBER CLUSTER” on page 420.

## 11.6 Data sharing lock contention analysis

In 11.5.3, “High INSERT data scenario” on page 398, we discuss a number of alternatives that can be used to solve global contention problems when doing a high number of inserts on multiple members of a data sharing group into a single table. Before you can decide which of these options to use, you first have to analyze the problem. For example, you have to determine which index the contention occurs on, and why it occurs, before you can make a change to the index design (making it random in this case).

In this section, we show how to analyze such a problem and how to zoom in on where the problem is. Once you understand the problem, it is much easier to solve the problem.

This scenario we investigate is similar to the high insert scenario, but the main focus in this section is to explain how to zoom in on the global contention problem. As before, a job is run on each member of the data sharing group, while a number of traces were active to capture some information about the run:

```
-STA TRACE (STAT )CLASS (1 3 4 5 )DEST (GTF )
-STA TRACE (ACCTG )CLASS (1 2 3 7 8 10 )DEST (GTF )
-STA TRACE (PERFM )CLASS (1 2 3 )DEST (GTF )
-STA TRACE (PERFM )CLASS (30 )DEST (GTF )
      IFCID (20 251 257 259 21 211 212 44 45 213 214 215 216 226 227 218 )
      TDATA (COR TRA CPU DIST )
```

Note that we used GTF as a trace destination. This is recommended when you do detailed tracing, as is the case here. Note that you have to run the GTF traces on all members of the data sharing group to capture all the information that is required for the analysis.

## 11.6.1 Analyzing the accounting information

Because we are only running a single job on each member, let us just look at an accounting trace report for one of the jobs (one accounting record for the job), just to show a different way to generate an accounting report. The counters are identical on all types of reports. The report was generated using the following OMEGAMON PE command stream:

```
DB2PM
GLOBAL
    TIMEZONE (+4)
    INCLUDE( DB2ID(D9C2)
            CORRNAME(DB2MEM*)
            IFCID (3 239)
        )
ACCOUNTING
TRACE
LAYOUT (LONG)
EXEC
```

As the trace data set is large, and to eliminate as many trace records as early as possible, only IFCID 3 (plan accounting) and 239 (package accounting) are selected for jobnames that start with DB2MEM.

An extract from the report is shown in Example 11-27. The job that ran on this member (D9C2) is called DB2MEM64. This job ran for 1 minute and 21 seconds, of which 1 minute and 12 seconds was spent inside DB2. Of the time in DB2 (class 2 time), 33 seconds were spent using CPU, 28 seconds were spent waiting inside DB2 for some event to complete, and 10 seconds was NOT ACCOUNT time, where we waited for the CPU (because the dispatching priority of our job is rather low).

**Example 11-27** Accounting information: plan level information

LOCATION: DB9C	OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)	PAGE: 1-1
GROUP: DB9CG	ACCOUNTING TRACE - LONG	REQUESTED FROM: NOT SPECIFIED
MEMBER: D9C2		TO: NOT SPECIFIED
SUBSYSTEM: D9C2		ACTUAL FROM: 08/19/09 20:42:16.26
DB2 VERSION: V9		

---- IDENTIFICATION -----			
ACCT TSTAMP: 08/19/09 20:42:16.26	PLANNAME: DSNREXX	WLM SCL: 'BLANK'	CICS NET: N/A
BEGIN TIME : 08/19/09 20:40:54.97	PROD TYP: N/P		CICS LUN: N/A
END TIME : 08/19/09 20:42:16.26	PROD VER: N/P	LWU NET: USIBMSC	CICS INS: N/A
REQUESTER : DB9C	<b>CORRNAME: DB2MEM64</b>	LWU LUN: SCPD9C2	
MAINPACK : DSNREXX	CORRNMBR: 'BLANK'	LWU INS: C4A9849D90E4	ENDUSER : 'BLANK'
PRIMAUTH : DB2R2	CONNTYPE: DB2CALL	LWU SEQ: 1	TRANSACTION: 'BLANK'
ORIGAUTH : DB2R2	CONNECT : DB2CALL		WSNAME : 'BLANK'

LOCATION: DB9C  
GROUP: DB9CG  
MEMBER: D9C2  
SUBSYSTEM: D9C2  
DB2 VERSION: V9

OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)  
ACCOUNTING TRACE - LONG

PAGE: 1-2  
REQUESTED FROM: NOT SPECIFIED  
TO: NOT SPECIFIED  
ACTUAL FROM: 08/19/09 20:42:16.26

```

----- IDENTIFICATION -----
ACCT TSTAMP: 08/19/09 20:42:16.26  PLANNAME: DSNREXX          WLM SCL: 'BLANK'      CICS NET: N/A
BEGIN TIME : 08/19/09 20:40:54.97  PROD TYP: N/P           CICS LUN: N/A
END TIME   : 08/19/09 20:42:16.26  PROD VER: N/P          CICS INS: N/A
REQUESTER  : DB9C                  CORRNAME: DB2MEM64      LUW LUN: SCPD9C2
MAINPACK   : DSNREXX              CORRMBR: 'BLANK'        LUW INS: C4A9849D90E4  ENDUSER : 'BLANK'
PRIMAUTH   : DB2R2                CONNTYPE: DB2CALL       LUW SEQ: 1             TRANSACT: 'BLANK'
ORIGAUTH   : DB2R2                CONNECT : DB2CALL       WSNAME  : 'BLANK'
  
```

TIMES/EVENTS	APPL (CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	ELAPSED TIME	EVENTS	HIGHLIGHTS
<b>ELAPSED TIME</b>	<b>1:21.28992</b>	<b>1:12.39805</b>	N/P	LOCK/LATCH(DB2+IRLM)	0.020552	33	THREAD TYPE : ALLIED
NONNESTED	32.017734	23.125865	N/A	SYNCHRON. I/O	3.148462	6525	TERM.CONDITION: NORMAL
STORED PROC	49.272186	49.272186	N/A	DATABASE I/O	2.188356	5557	INVOKE REASON : DEALLOC
UDF	0.000000	0.000000	N/A	LOG WRITE I/O	0.960106	968	COMMITTS : 10001
TRIGGER	0.000000	0.000000	N/A	OTHER READ I/O	0.000574	1	ROLLBACK : 0
				OTHER WRTE I/O	0.001254	2	SVPT REQUESTS : 0
CP CPU TIME	39.796018	<b>33.643926</b>	N/P	SER.TASK SWTCH	17.070168	10007	SVPT RELEASE : 0
AGENT	39.796018	33.643926	N/A	<b>UPDATE COMMIT</b>	<b>16.365151</b>	<b>9976</b>	SVPT ROLLBACK : 0
NONNESTED	10.919427	4.767335	N/P	OPEN/CLOSE	0.305012	12	INCREM.BINDS : 0
STORED PROC	28.876591	28.876591	N/A	SYSLGRNG REC	0.052003	4	UPDATE/COMMIT : 1.00
UDF	0.000000	0.000000	N/A	EXT/DEL/DEF	0.034224	10	SYNCH I/O AVG.: 0.000483
TRIGGER	0.000000	0.000000	N/A	OTHER SERVICE	0.313778	5	PROGRAMS : 7
PAR.TASKS	0.000000	0.000000	N/A	ARC.LOG(QUIES)	0.000000	0	MAX CASCADE : 2
				LOG READ	0.000000	0	PARALLELISM : NO
SECP CPU	0.000000	N/A	N/A	DRAIN LOCK	0.000000	0	
				CLAIM RELEASE	0.000000	0	
SE CPU TIME	0.000000	0.000000	N/A	PAGE LATCH	0.000455	5	
NONNESTED	0.000000	0.000000	N/A	NOTIFY MSGS	0.000000	0	
STORED PROC	0.000000	0.000000	N/A	<b>GLOBAL CONTENTION</b>	<b>7.851154</b>	<b>1211</b>	
UDF	0.000000	0.000000	N/A	COMMIT PH1 WRITE I/O	0.000000	0	
TRIGGER	0.000000	0.000000	N/A	ASYNCH CF REQUESTS	0.435313	71	
				TCP/IP LOB	0.000000	0	
PAR.TASKS	0.000000	0.000000	N/A	TOTAL CLASS 3	28.527932	17855	
<b>SUSPEND TIME</b>	0.000000	<b>28.527932</b>	N/A				
AGENT	N/A	28.527932	N/A				
PAR.TASKS	N/A	0.000000	N/A				
STORED PROC	0.000000	N/A	N/A				
UDF	0.000000	N/A	N/A				
NOT ACCOUNT.	N/A	10.226194	N/A				
DB2 ENT/EXIT	N/A	40500	N/A				
EN/EX-STPROC	N/A	0	N/A				
EN/EX-UDF	N/A	0	N/A				
DCAPT.DESCR.	N/A	N/A	N/P				
LOG EXTRACT.	N/A	N/A	N/P				

GLOBAL	CONTENTION	L-LOCKS	ELAPSED TIME	EVENTS	GLOBAL	CONTENTION	P-LOCKS	ELAPSED TIME	EVENTS
L-LOCKS			0.010948	7	<b>P-LOCKS</b>			7.840206	1204
PARENT (DB,TS,TAB,PART)			0.000000	0	PAGESET/PARTITION			0.040448	9
CHILD (PAGE,ROW)			0.009172	6	<b>PAGE</b>			<b>7.584542</b>	<b>1170</b>
OTHER			0.001776	1	OTHER			0.215217	25

SQL DML	TOTAL
SELECT	130000
<b>INSERT</b>	<b>10000</b>
UPDATE	0
MERGE	0
DELETE	1

DESCRIBE	41
DESC.TBL	0
PREPARE	41
OPEN	41
FETCH	83
CLOSE	41

DML-ALL 140248

GROUP TOT4K TOTAL

-----	-----
GBP-DEPEND GETPAGES	91501
READ(XI)-DATA RETUR	11995
READ(XI)-NO DATA RT	4013
READ(NF)-DATA RETUR	207
READ(NF)-NO DATA RT	1499
PREFETCH PAGES READ	0
CLEAN PAGES WRITTEN	0
UNREGISTER PAGE	0
ASYNCH GBP REQUESTS	0
EXPLICIT X-INVALID	0
ASYNCH SEC-GBP REQ	0
PG P-LOCK LOCK REQ	29535
SPACE MAP PAGES	4
DATA PAGES	833
INDEX LEAF PAGES	28698
PG P-LOCK UNLOCK REQ	30094
<b>PG P-LOCK LOCK SUSP</b>	<b>1178</b>
SPACE MAP PAGES	0
DATA PAGES	3
<b>INDEX LEAF PAGES</b>	<b>1175</b>
WRITE AND REGISTER	39899
WRITE & REGISTER MULT	590
CHANGED PAGES WRITTEN	40746
WRITE TO SEC-GBP	N/A

---

In this section, we focus on the suspension time (28 seconds). This mainly consists of a little bit of database I/O wait time, 2 seconds, 16 seconds of update commit time, and 7.8 seconds of global contention time.

Before we zoom in on the global contention time, let us discuss the update commit time. This value is high because the workload commits after each call to a stored procedure. As a call to the ADDEMP SP adds only a single employee and then commits, and the run inserts 10,000 employees, the update commit time is high. Although when you look at the update commit time per event ( $16.365151 / 9976$ ), it is only 1.64 msec. which is certainly acceptable. In a real life environment, you normally insert multiple rows before committing, so it would be an easy change if we wanted to reduce the update commit time.

Now let us return to the topic of the 7.8 seconds of global contention time. When you look at the drill down values of the counter in Example 11-27 on page 404, page P-lock suspension time is the major contributor by far, with 7.5 seconds. When we look at the GBP section, these page P-lock suspensions are almost all for index leaf pages.

It looks like we have a “hot” index. Now we need to find which index, or indexes, are causing most of the page P-lock suspensions.



We also have a quick look at the package level accounting information to see if we can identify which package is responsible for most of the global lock contention. An excerpt is shown in Example 11-28. It looks like the EMPADD stored procedure is responsible for the page P-lock contention. (Note that there is no GBP activity information at the package level.)

**Example 11-28** Accounting information: package level information

EMPADD	VALUE	EMPADD	TIMES	EMPADD	TIME	EVENTS	TIME/EVENT
-----	-----	-----	-----	-----	-----	-----	-----
TYPE	PACKAGE	ELAPSED TIME - CL7	21.632970	LOCK/LATCH	0.012386	18	0.000688
LOCATION	DB9C	CP CPU TIME	7.805586	SYNCHRONOUS I/O	2.207015	5369	0.000411
COLLECTION ID	RGDB07	AGENT	7.805586	OTHER READ I/O	0.000000	0	N/C
PROGRAM NAME	EMPADD	PAR.TASKS	0.000000	OTHER WRITE I/O	0.000000	0	N/C
CONSISTENCY TOKEN	18952F0501AACB52	SE CPU TIME	0.000000	SERV.TASK SWITCH	0.125954	10	0.012595
ACTIVITY TYPE	NATIVE SQL PROC	SUSPENSION-CL8	10.107145	ARCH.LOG(QUIESCE)	0.000000	0	N/C
ACTIVITY NAME	EMPADD	AGENT	10.107145	ARCHIVE LOG READ	0.000000	0	N/C
SCHEMA NAME	RGDB07	PAR.TASKS	0.000000	DRAIN LOCK	0.000000	0	N/C
NBR OF ALLOCATIONS	60000	NOT ACCOUNTED	3.720239	CLAIM RELEASE	0.000000	0	N/C
SQL STMT - AVERAGE	2.00			PAGE LATCH	0.000380	3	0.000127
SQL STMT - TOTAL	120000	CP CPU SU	209546	NOTIFY MESSAGES	0.000000	0	N/C
SUCC AUTH CHECK	YES	AGENT	209546	<b>GLOBAL CONTENTION</b>	<b>7.761410</b>	1192	0.006511
		PAR.TASKS	0	TCP/IP LOB	0.000000	0	N/C
		SE CPU SU	0	TOTAL CL8 SUSPENS.	10.107145	6592	0.001533
		DB2 ENTRY/EXIT	20000				
EMPADD	ELAPSED TIME	EVENTS	EMPADD	ELAPSED TIME	EVENTS		
-----	-----	-----	-----	-----	-----		
GLOBAL CONTENTION L-LOCKS	0.000000	0	GLOBAL CONTENTION P-LOCKS	7.761410	1192		
PARENT (DB,TS,TAB,PART)	0.000000	0	PAGESET/PARTITION	0.018211	4		
CHILD (PAGE,ROW)	0.000000	0	<b>PAGE</b>	<b>7.542204</b>	1167		
OTHER	0.000000	0	OTHER	0.200995	21		
EMPADD	TOTAL						
-----	-----						
SELECT	20000						
<b>INSERT</b>	<b>10000</b>						
UPDATE	0						
DELETE	0						
DESCRIBE	0						
PREPARE	0						
OPEN	0						
FETCH	0						
CLOSE	0						
LOCK TABLE	0						
CALL	50000						
EMPADD	TOTAL						
-----	-----						
BPOOL HIT RATIO (%)	96						
GETPAGES	150229						
BUFFER UPDATES	40938						
SYNCHRONOUS WRITE	0						
SYNCHRONOUS READ	5263						
SEQ. PREFETCH REQS	0						
LIST PREFETCH REQS	0						
DYN. PREFETCH REQS	0						
PAGES READ ASYNCHR.	0						

## 11.6.2 Lock suspension report

The lock suspension report was discussed in great detail in “Lock suspension information” on page 265 for a non-data sharing environment.

The report is identical in a data sharing environment. However, you can generate a report using the SCOPE(GROUP) subcommand, and provide the trace data sets from all members as input. This command is as follows:

```
DB2PM
  GLOBAL
    TIMEZONE (+4)
  LOCKING
    REPORT
      SCOPE(GROUP)
      LEVEL(SUSPENSION)
      ORDER(DATABASE-PAGESET)
EXEC
```

An excerpt of the lock suspension report is shown in Example 11-29. Note that the report has an additional level of granularity, the DB2 member. The example shows the locks for database RGDB07 table space GLWSEMP: First, the lock suspensions for member D9C1, followed by the lock suspensions on member D9C2, and a “\*group total\*” for the resource. In this case, we have seven lock suspensions on D9C1 and 13 on D9C2, 20 for the group on RGDB07.GLWSEMP. Note that most of them are global contention.

*Example 11-29 SCOPE(GROUP) lock suspension report*

LOCATION: DB9C GROUP: DB9CG		OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2) LOCKING REPORT - SUSPENSION						PAGE: 1-3 REQUESTED FROM: NOT SPECIFIED TO: NOT SPECIFIED INTERVAL FROM: 08/19/09 20:40:55.09 TO: 08/19/09 20:43:57.71								
DB2 VERSION: V9		ORDER: DATABASE-PAGESET SCOPE: GROUP														
DATABASE PAGESET MEMBER		--- L O C K R E S O U R C E ---		TOTAL	LOCAL	GLOB.	S.NFY	----	NORMAL	----	TIMEOUT/CANCEL	----	R E A S O N S	----	DEADLOCK	----
		TYPE	NAME	SUSPENDS	LATCH	IRLMQ	OTHER	NMBR	AET	NMBR	AET	NMBR	AET	NMBR	AET	
-----																
RGDB07																
GLWSEMP																
D9C1																
	DATAPAGE	PAGE=X'3005F0'		5	0	5	0	5	0.001258	0	N/C	0	N/C			
	P/P PLCK	PART= 3 PAGE=X'000200' BPID=BP2		1	0	1	0	1	0.000142	0	N/C	0	N/C			
	PART SPL	N/P		1	0	0	0	1	0.000501	0	N/C	0	N/C			
	** SUM OF	D9C1	**	7	0	6	0	7	0.000990	0	N/C	0	N/C			
D9C2																
	DATAPAGE	PAGE=X'200049'		1	0	0	0	1	0.000622	0	N/C	0	N/C			
	DATAPAGE	PAGE=X'30041C'		6	0	6	0	6	0.001529	0	N/C	0	N/C			
	MASSDEL	N/P		2	0	0	0	2	0.000997	0	N/C	0	N/C			
	P/P CAST	PART= 4 BPID=BP2		1	0	1	0	1	0.032722	0	N/C	0	N/C			
	P/P PLCK	PART= 3 PAGE=X'000200' BPID=BP2		1	0	1	0	1	0.000707	0	N/C	0	N/C			
	P/P PLCK	PART= 4 PAGE=X'000200' BPID=BP2		2	0	2	0	2	0.004090	0	N/C	0	N/C			
	** SUM OF	D9C2	**	13	0	10	0	13	0.004107	0	N/C	0	N/C			
*GROUP TOTAL*																
GLWSEMP				20	0	16	0	20	0.003016	0	N/C	0	N/C			
					4	0	0									

To make the information easier to digest, we use an edit macro that searches the \*group total\* lines and the pageset names. The result is shown in Example 11-30. The report shows that the majority of the lock suspensions are global lock suspensions on index GLWXEMP3. The other lock contentions are more likely page latch contentions of a second transaction waiting behind a page latch held by the first transaction, where the first transaction is waiting for a page P-lock.

*Example 11-30 Edited lock suspension report*

LOCATION: DB9C		OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)									
GROUP: DB9CG		LOCKING REPORT - SUSPENSION									
		ORDER: DATABASE-PAGESET									
DB2 VERSION: V9		SCOPE: GROUP									
DATABASE		--SUSPEND REASONS-- ----- R E S U M E									
PAGESET	---	L O C K	R E S O U R C E	---	TOTAL	LOCAL	<b>GLOB.</b>	S.NFY	----	NORMAL	----
MEMBER		TYPE	NAME		SUSPENDS	LATCH	IRLMQ	<b>OTHER</b>	NMBR		AET
-----											
*GROUP TOTAL*											
'BLANK'					234	0	34	7	234		0.001487
						200	0	0			
-----											
*GROUP TOTAL*											
GLWSEMP					20	0	16	0	20		0.003016
						4	0	0			
-----											
*GROUP TOTAL*											
GLWXEMP2					23	0	18	0	23		0.033771
						1	0	4			
-----											
*GROUP TOTAL*											
<b>GLWXEMP3</b>					1767	0	<b>1365</b>	0	1767		0.004128
						2	0	<b>400</b>			
-----											

Looking at the details for GLWXEMP3 (bringing back some of the “excluded” lines of the report, as shown in Example 11-31), it are indeed page P-locks that are responsible for the global contention, and most likely page latches causing the “other” suspensions.

**Example 11-31 Lock suspensions or GLWXEMP3**

LOCATION: DB9C GROUP: DB9CG		OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2) LOCKING REPORT - SUSPENSION							PAGE: 1-6 REQUESTED FROM: NOT SPECIFIED TO: NOT SPECIFIED INTERVAL FROM: 08/19/09 20:40:55.09 TO: 08/19/09 20:43:57.71				
DB2 VERSION: V9		ORDER: DATABASE-PAGESET SCOPE: GROUP											
DATABASE PAGESET MEMBER		--- L O C K R E S O U R C E ---		TOTAL	--SUSPEND REASONS--	----- R E S U M E		R E A S O N S -----					
		TYPE	NAME	SUSPENDS	LOCAL LATCH	GLOB. IRLMQ	S.NFY OTHER	----- NMNR	NORMAL --- AET	TIMEOUT/CANCEL NMNR	--- DEADLOCK --- AET		
-----													
...													
GLWXEMP3 D9C1		PAGE	PAGE=X'3000EA'	2	0	0	0	2	0.000043	0	N/C	0	N/C
			BPID=BP3		0	0	2						
		PAGE	PAGE=X'3000EC'	1	0	0	0	1	0.000036	0	N/C	0	N/C
			BPID=BP3		0	0	1						
		PAGE	PAGE=X'3000EE'	7	0	0	0	7	0.000045	0	N/C	0	N/C
			BPID=BP3		0	0	7						
		PAGE	PAGE=X'3000EF'	2	0	0	0	2	0.000042	0	N/C	0	N/C
			BPID=BP3		0	0	2						
		PAGE	PAGE=X'3000E8'	8	0	0	0	8	0.000044	0	N/C	0	N/C
			BPID=BP3		0	0	8						
		PAGE	PAGE=X'3000F1'	4	0	0	0	4	0.000043	0	N/C	0	N/C
			BPID=BP3		0	0	4						
		PAGE	PAGE=X'3000F3'	5	0	0	0	5	0.000102	0	N/C	0	N/C
			BPID=BP3		0	0	5						
		PAGE	PAGE=X'3000F5'	2	0	0	0	2	0.000086	0	N/C	0	N/C
			BPID=BP3		0	0	2						
		PAGEPLCK	PART= 4	24	0	24	0	24	0.001714	0	N/C	0	N/C
			PAGE=X'0000EA'		0	0	0						
			BPID=BP3										
		PAGEPLCK	PART= 4	20	0	20	0	20	0.003008	0	N/C	0	N/C
			PAGE=X'0000EC'		0	0	0						
			BPID=BP3										
		PAGEPLCK	PART= 4	19	0	19	0	19	0.002261	0	N/C	0	N/C
			PAGE=X'0000EE'		0	0	0						
			BPID=BP3										
....													

For this problem, we probably have enough information to understand what the problem is. We are dealing with inserts that are causing page P-lock contention on index leaf pages. Looking at the index definition, we find that the key column is EMP\_NO in ascending order (see Example 11-32).

**Example 11-32 GLWXEMP3 definition**

DB2 Admin -- D9C1 Columns in Index RGDB07.GLWXEMP3

Seq	Column Name	No	0	Col	Type	Length	Sca
*		*	*	*		*	
---	---	---	---	---	---	---	---
	EMP_NO	1	A	INTEGER		4	

Because we know that the ADDEMP stored procedure always generates a new employee number that is one higher than the last one, and with an ascending index on EMP\_NO, all index keys go into the same page. Because we run an insert job that is adding employees on both members, the last index leaf page is constantly being switched back and forth between both members, using a page P-lock to track who is holding the page at any one point in time.

### 11.6.3 Analyzing an individual global lock suspension

However, let us assume that the problem is not just the number of global contentions, as is the case in 11.6.2, “Lock suspension report” on page 408, but a long global contention. In that case, the analysis would be the same up to this point. In the lock suspension report, you would be looking for a long AET lock suspension, such as:

```
----- R E S U M E   R E A S O N S -----
---- NORMAL ----  TIMEOUT/CANCEL  --- DEADLOCK ---
NMBR      AET  NMBR      AET  NMBR      AET
-----
  22    2.001841    0      N/C    0      N/C

  24    0.003535    0      N/C    0      N/C
```

Once you find the resource with the long average suspension time, you can list all suspensions for that resource using the following OMEGAMON PE input command:

```
GLOBAL
  TIMEZONE (+4)
  LOCKING
    TRACE
      INCLUDE (DATABASE (RGDB07) PAGESET (GLWXEMP3))
      LEVEL (SUSPENSION)
EXEC
```

An excerpt of the report is shown in Example 11-33. The report conveniently shows the amount of time the transactions were suspended in the SUSP.TIME column. If the report is rather long, it is probably easiest to perform a sort on the suspension time to find the longest ones.

**Example 11-33** Lock suspension trace for an object

```

....
LOCATION: DB9C                                OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)
GROUP: DB9CG                                LOCKING TRACE - SUSPENSION
MEMBER: D9C2                                PAGE: 1-351
SUBSYSTEM: D9C2                             REQUESTED FROM: NOT SPECIFIED
DB2 VERSION: V9                             TO: NOT SPECIFIED
PRMAUTH CORRNAME CONNTYPE                  ACTUAL FROM: 08/19/09 20:40:56.10
ORIGAUTH CORRNMBR INSTANCE                 PAGE DATE: 08/19/09
PLANNAME CONNECT                          SCOPE: MEMBER

EVENT TIMESTAMP    --- L O C K   R E S O U R C E ---
RELATED TIMESTAMP  TYPE      NAME      EVENT SPECIFIC DATA
-----
DB2R2  DB2MEM64  DB2CALL  20:41:47.79660703 LOCK  PAGEPLCK  DB  =RGDB07  SUSP.TIME =0.000751  LOCAL CONTENTION=N
DB2R2  'BLANK'   C4A9849D90E4  20:41:47.79585629 RESUME                OB  =GLWXEMP3  DURATION  =INTEREST  LATCH CONTENTION=N
DSNREXX DB2CALL                                PART= 4        STATE      =X        IRLM QUEUED REQ =N
                                           PAGE=X'0000D5' RESUME RSN=NORMAL  GLOBAL CONT.  =Y*
                                           BPID=BP3       XES PROP  =Y        NOTIFY MSG SENT =N
                                           XES FORC  =Y        FALSE/SYNC-ASYN
                                           XES ASYN  =Y        RETAINED LOCK  =N
                                           MODIFY GLOBAL P-LOCK
                                           HASH      =X'04314CD5'

                20:41:47.86256073 LOCK  PAGEPLCK  DB  =RGDB07  DURATION=INTEREST STATE=X      XES PROP=Y
                RELATED TIMESTAMP    SUSPEND                OB  =GLWXEMP3  ORIG.RSN=INTER SYSTEM      XES FORC=Y
                EVENT                PART= 4        MODIFY GLOBAL P-LOCK      XES ASYN=Y
                TYPE                  PAGE=X'0000D5'
                NAME                  BPID=BP3        HASH      =X'04314CD5'

                20:41:48.39473015 LOCK  PAGEPLCK  DB  =RGDB07  SUSP.TIME =0.532169  LOCAL CONTENTION=N
                20:41:47.86256073 RESUME                OB  =GLWXEMP3  DURATION  =INTEREST  LATCH CONTENTION=N
                EVENT                PART= 4        STATE      =X        IRLM QUEUED REQ =N
                TYPE                  PAGE=X'0000D5' RESUME RSN=NORMAL  GLOBAL CONT.  =Y*
                NAME                  BPID=BP3       XES PROP  =Y        NOTIFY MSG SENT =N
                                           XES FORC  =Y        IRLM GLOBAL CONT.
                                           XES ASYN  =Y        RETAINED LOCK  =N
                                           MODIFY GLOBAL P-LOCK
                                           HASH      =X'04314CD5'

                20:41:48.39712356 LOCK  PAGEPLCK  DB  =RGDB07  DURATION=INTEREST STATE=X      XES PROP=Y
                20:41:48.39712356 SUSPEND                OB  =GLWXEMP3  ORIG.RSN=INTER SYSTEM      XES FORC=Y
                EVENT                PART= 4        MODIFY GLOBAL P-LOCK      XES ASYN=Y
                TYPE                  PAGE=X'0000D5'
                NAME                  BPID=BP3        HASH      =X'04314CD5'

                20:41:48.40448737 LOCK  PAGEPLCK  DB  =RGDB07  SUSP.TIME =0.007364  LOCAL CONTENTION=N
                20:41:48.39712356 RESUME                OB  =GLWXEMP3  DURATION  =INTEREST  LATCH CONTENTION=N
                EVENT                PART= 4        STATE      =X        IRLM QUEUED REQ =N
                TYPE                  PAGE=X'0000D5' RESUME RSN=NORMAL  GLOBAL CONT.  =Y*
                NAME                  BPID=BP3       XES PROP  =Y        NOTIFY MSG SENT =N
                                           XES FORC  =Y
                                           XES ASYN  =Y
                                           MODIFY GLOBAL P-LOCK
                                           HASH      =X'04314CD5'
...

```

We discover one in the report (resumed at 20:41:48.39473015) where the transaction has been waiting for a page P-lock on page x'D5' in partition 4 of the GLWXEMP3 index for 0.532169 seconds. Note that the right hand side of the report confirms that this is a global contention, and that it was IRLM global contention (not XES contention, or a false contention/heuristic converted request).

Let us now have a closer look at the lock request that was suspended. We use a RECTRACE with a FROM/TO clause to limit the amount of data produced by the report, as follows:

```

GLOBAL
  TIMEZONE(+4)
  FROM(,20:41:47.86)
  TO(,20:41:48.42)
  INCLUDE (
    DB2ID(D9C2)
  )

```

```

RECTRACE
TRACE
LEVEL(LONG)
EXEC

```

An excerpt of the output is shown in Example 11-34, where:

1. IFCID 44 shows the page P-lock request that got suspended. It is a request for an exclusive page P-lock for page x'D5' of index GLWXEMP3.
2. At 20:41:48.39473015, the lock request is (normally) resumed (IFCID 45), and the reason for the suspension is global IRLM contention, which means the page P-lock was held on another member.
3. This is followed by an IFCID 21 trace record providing the information about the end of the lock request, including the IRLM return code and subcodes (reason codes). In this case, the request completed successfully (RC 0).
4. Because this was a page P-lock request and IFCID 259 was traced, there is also a buffer manager trace record confirming that the page P-lock for the index page is now held in an exclusive state by this member (D9C2).

**Example 11-34** *Suspended page P-lock request*

```

...
                20:41:47.86256073 944105 1 44 LOCK SUSPEND 'BLANK'
                24.52191506                                NETWORKID: USIBMSC LUNAME: SCPD9C2 LUWSEQ: 1
-----
LOCK RES TYPE: PAGE P-LOCK          DBID: RGD807  OBID: GLWXEMP3  PART: 4 BPID: 3  PAGE: X'0000D5'
IRLM FUNC CODE: LOCK (NAME)        STATE: EXCLUSIVE  DURATION: INTEREST  REASON SUSP: IS
REQ TOKEN: X'00000000'  LOCK ATTRIBUTES: P-LOCK GLOBAL MODIFY  FORCE  PROP TO XES: YES  ASYN TO XES: YES
PARENT TOKEN: X'00000000'  LOCK HASH VALUE: X'04314CD5'
QW0044CL: X'00'  QW0044FL: X'13'
-----
LOCATION: DB9C                                OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)                                PAGE: 1-20
GROUP: DB9CG                                RECORD TRACE - LONG                                REQUESTED FROM: ALL 20:41:47.86
MEMBER: D9C2                                TO: DATES 20:41:48.42
SUBSYSTEM: D9C2                                ACTUAL FROM: 08/19/09 20:41:47.86
DB2 VERSION: V9                                PAGE DATE: 08/19/09
PRIMAUTH CONNECT  INSTANCE  END_USER  WS_NAME  TRANSACT
ORIGAUTH CORRNAME CONNTYPE  RECORD TIME  DESTNO ACE IFC  DESCRIPTION  DATA
PLANNAME CORRNMBR  TCB CPU TIME  ID
-----
DB2R2  DB2CALL  C4A9849D90E4 'BLANK'  'BLANK'  'BLANK'
DB2R2  DB2MEM64 DB2CALL  20:41:48.39473015 944106 1 45 LOCK <-- NETWORKID: USIBMSC LUNAME: SCPD9C2 LUWSEQ: 1
DSNREXX 'BLANK'  24.52194042  RESUME  REASON FOR RESUME : NORMAL RESUME
REASON FOR SUSPEND : X'08'
IRLM LATCH CONTENTION : NO
IRLM QUEUED REQUEST : NO
LOCAL RESOURCE CONTENTION : NO
RETAINED LOCK CONTENTION : NO
GLOBAL RESOURCE CONTENTION : YES
INTER-SYSTEM MESSAGE SENDING : NO
GLOBAL CONTENTION EXTENT : X'F0'
XES GLOBAL CONTENTION : NO
IRLM GLOBAL CONTENTION : YES
FALSE CONTENTION : NO
QW0045W4 NO QW0045W6 NO QW0045W8 NO
QW0045X1 YES QW0045X2 YES QW0045X5 NO
QW0045X6 NO QW0045X7 NO QW0045X8 NO
                20:41:48.39476005 944107 1 21 LOCK DETAIL 'BLANK'
                24.52195109                                NETWORKID: USIBMSC LUNAME: SCPD9C2 LUWSEQ: 1
-----
LOCK RES TYPE: PAGE P-LOCK          DBID: RGD807  OBID: GLWXEMP3  PART: 4 BPID: 3  PAGE: X'0000D5'
IRLM FUNC CODE : LOCK (NAME)        RETURN TOKEN: X'7F6A9260'  REQUEST TOKEN : X'00000000'
LOCK STATE : EXCLUSIVE  DB2 TOKEN : X'00A7100021928428'  IRLM RETURN CODE : 0
LOCK ATTRIBUTES: MODIFY FORCE  PROP TO XES : YES  ASYN TO XES : YES
LOCK DURATION : INTEREST  REQUEST TYPE: P-LOCK  IRLM RETURN SUBCODE: B'0000000000000000'
PARENT TOKEN : X'00000000'  GLOBAL/LOCAL: GLOBAL  OWNER : 'BLANK'
LOCK HASH VALUE : X'04314CD5'
QW0021CL: X'00'  QW0021U : X'00E90000203623E8'  QW0021CT: X'00000000'  QW0021FL: B'00010011'
QW0021F3: B'11110000'  QW0021O : X'00E92EB2EB2EB280'  QW0021IR: X'0000'  QW0021F2: B'00000000'
-----
                20:41:48.39477084 944108 1 259 BUFFER MANAGER 'BLANK'
                24.52195943  PG P-LOCK REQ NETWORKID: USIBMSC LUNAME: SCPD9C2 LUWSEQ: 1
-----

```

```

P-LOCK TYPE      : PAGE          DBID: RGDB07      OBID: GLWXEMP3      PARTITION NMBR : 4      BP ID   : X'03'
IRLM FUNC CODE   : LOCK          OBJECT TYPE : INDEX PAGE  MODIFY          : YES      PAGE NMBR: X'0000D5'
DB2 MEMBER NAME  : 'BLANK'       CONDITIONAL: NO        RESTART: NO       REQUESTED STATE: EXCLUSIVE
OLD HELD STATE   : NOT HELD      NEW HELD STATE : EXCLUSIVE
QW0259EV X'D3'   QW0259TK X'00000000' QW0259RC X'00000000' QW0259RS X'00000000' QW0259PC X'0000'
-----

```

Going back in time a little bit, the trace shows that this suspension was part of an insert statement, as shown in Example 11-35.

#### Example 11-35 Insert statement

```

SUBSYSTEM: D9C2                                     ACTUAL FROM: 08/19/09 20:41:47.86
DB2 VERSION: V9                                     PAGE DATE: 08/19/09
PRIMAUTH CONNECT INSTANCE      END_USER      WS_NAME      TRANSACT
ORIGAUTH CORRNAME CONNTYPE      RECORD TIME  DESTNO ACE IFC DESCRIPTION  DATA
PLANNAME CORRNMBR      TCB CPU TIME  ID
-----
DB2R2  DB2CALL  C4A9849D90E4 'BLANK'      'BLANK'      'BLANK'
DB2R2  DB2MEM64 DB2CALL      20:41:47.86231287 944097 1 61 INSERT  ---> NETWORKID: USIBMSC  LUNAME: SCPD9C2  LUWSEQ: 1
DSNREXX 'BLANK'      24.52173267      START
                                           LOCATION NAME : DB9C
                                           PKG COLLECTION ID : RGDB07
                                           PROGRAM NAME : EMPADD
                                           CONSISTENCY TOKEN : X'18952F0501AACB52'
                                           STATEMENT NUMBER : 1
                                           STATEMENT TYPE : INSERT TYPE
                                           CURSOR NAME : N/P
                                           QUERY COMMAND ID : N/P
                                           QUERY INSTANCE ID : N/P
                                           ISOLATION : CS
                                           REOPTIMIZATION : NO

```

This is what happened to the transaction that was suspended. Because this was a global contention suspension, this means that the holder of the lock was on another member. As we are only using two members during our test, we know that the hold must be held on D9C1.

To see what happens on that member, we run a record trace for a similar time frame than we used on D9C2, and do a search in the output for this resource (page x'0000D5') using the following command input:

```

DB2PM
GLOBAL
TIMEZONE(+4)
FROM(,20:41:47.86)
TO(,20:41:48.42)
INCLUDE (
    DB2ID(D9C1)
)
RECTRACE
TRACE
LEVEL(LONG)
EXEC

```



The relevant portion of the report is shown in Example 11-36.

**Example 11-36** P-lock holder on D9C1

LOCATION: DB9C				OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4R2)				PAGE: 1-2			
GROUP: DB9CG				RECORD TRACE - LONG				REQUESTED FROM: ALL 20:41:47.86			
MEMBER: D9C1								TO: DATES 20:41:48.42			
SUBSYSTEM: D9C1								ACTUAL FROM: 08/19/09 20:41:47.86			
DB2 VERSION: V9								PAGE DATE: 08/19/09			
PRIMAUTH CONNECT		INSTANCE	END USER	WS NAME			TRANSACTION				
ORIGAUTH CORRNAME		CONNTYPE	RECORD TIME	DESTNO ACE	IFC	DESCRIPTION	DATA				
PLANNAME CORRNMBR			TCB CPU TIME	ID							
-----											
DB2R2	DB2CALL	C4A9849D00D8	'BLANK'	'BLANK'							
DB2R2	DB2MEM63	DB2CALL	20:41:47.86050840	947654	1	21	LOCK DETAIL	NETWORKID:	USIBMSC	LUNAME: SCPD9C1 LUWSEQ: 1	
DSNREXX	'BLANK'	24.61152305									
-----											
LOCK RES TYPE: PAGE P-LOCK				DBID: 304	OBID: 16	PART: 4		BPID: 3	PAGE: X'0000D5'		
IRLM FUNC CODE : LOCK (NAME)				RETURN TOKEN: X'7F6AE120'		REQUEST TOKEN :		X'00000000'			
LOCK STATE : EXCLUSIVE				DB2 TOKEN : X'00A7F00021733428'		IRLM RETURN CODE :		0			
LOCK ATTRIBUTES: MODIFY FORCE				PROP TO XES : YES		ASYN TO XES :		NO			
LOCK DURATION : INTEREST				REQUEST TYPE: P-LOCK		IRLM RETURN SUBCODE: B'0000000000000000'					
PARENT TOKEN : X'00000000'				GLOBAL/LOCAL: GLOBAL		OWNER :		'BLANK'			
CACHED STATE : N/A						LOCK HASH VALUE :		X'04314CD5'			
QW0021CL: X'00'		QW0021U : X'010800001FED7F20'		QW0021CT: X'00000000'		QW0021FL: B'00010011'					
QW0021F3: B'10000000'		QW00210 : X'0108127463927430'		QW0021IR: X'0000'		QW0021F2: B'00000000'					
-----											
20:41:47.86051431				947655	1	259	BUFFER MANAGER 'BLANK'				
24.61152543				PG P-LOCK REQ NETWORKID: USIBMSC LUNAME: SCPD9C1 LUWSEQ: 1							
-----											
P-LOCK TYPE : PAGE		DBID: 304		OBID: 16		PARTITION NMBR : 4		BP ID :	X'03'		
IRLM FUNC CODE : LOCK		OBJECT TYPE : INDEX PAGE		MODIFY : YES		PAGE NMBR: X'0000D5'					
DB2 MEMBER NAME : 'BLANK'		CONDITIONAL: NO		RESTART: NO		REQUESTED STATE: EXCLUSIVE					
OLD HELD STATE : NOT HELD						NEW HELD STATE : EXCLUSIVE					
QW0259EV X'D3'		QW0259TK	X'00000000'	QW0259RC	X'00000000'	QW0259RS	X'00000000'	QW0259PC	X'0000'		
-----											
...											
...											
SYSOPR	D9C1	C4A940C12F33	'BLANK'	'BLANK'	'BLANK'						
SYSOPR	020.TLPL	'BLANK'	20:41:48.39418215	947661	2	259	BUFFER MANAGER	NETWORKID: USIBMSC	LUNAME: SCPD9C1 LUWSEQ: 1		
'BLANK'	KP1E	1.74118468 PG P-LOCK REQ									
-----											
P-LOCK TYPE : PAGE		DBID: 304		OBID: 16		PARTITION NMBR : 4		BP ID :	X'03'		
IRLM FUNC CODE : CHANGE FROM P-LOCK EXIT		OBJECT TYPE : INDEX PAGE		MODIFY : NO		PAGE NMBR: X'0000D5'					
DB2 MEMBER NAME : D9C2		CONDITIONAL: NO		RESTART: NO		REQUESTED STATE: EXCLUSIVE					
OLD HELD STATE : EXCLUSIVE						NEW HELD STATE : NOT HELD					
QW0259EV X'D7'		QW0259TK	X'00000000'	QW0259RC	X'00000000'	QW0259RS	X'00000000'	QW0259PC	X'0000'		
-----											

On member D9C,1 the page P-lock was obtained at 20:41:47.86050840. (Note that the DBID and OBID are not translated into regular names. This is because the trace data set wrapped around on D9C1 and we lost the IFCID 105 information that handles the OBID translation from numbers to names. But rest assured that DBID 304 is indeed RGDB07 and OBID 16 is GLWXEMP3.)

There is no unlock of the page P-lock on D9C1 until, at 20:41:48.39418215, the P-lock exit obtains the page P-lock on behalf of member D9C2. When member D9C2 tries to obtain the page P-lock for this page and gets suspended because it is not available, as part of the P-lock negotiation, a P-lock exit is scheduled on the other member (D9C1) that is holding the page P-lock. Before the page P-lock can be obtained, if there were updates to the index page (which is highly likely, because every member is inserting into it), the index updates have to be forced to the active log data set and the index page has to be written to the group buffer pool before the page P-lock is obtained.

At 20:41:48.39418215, all of that work is done and the page P-lock is obtained. The trace was not detailed enough to allow us to determine whether or not logging and writing to the GBP had to take place.

Note that:


- The IFCID 259 record is written by the P-lock exit, 020.TLPLKP1E. This is a system agent. If you only trace a specific plan name or auth ID, this trace record will not be produced.
- The DB2 MEMBER NAME is D9C2, which is the member requesting the P-lock.
- The OLD HELD STATE is exclusive. D9C1 was holding the page P-lock in exclusive mode.
- The NEW HELD STATE is not held. This is the state of the page P-lock on this member (member D9C1 at the end of the P-lock negotiation). D9C1 gives up the page P-lock.
- The REQUESTED STATE is the state the page P-lock is requested by the P-lock exit on behalf of the other member (so this is really the state for which the other member is requesting the P-lock).

Going back in time a bit more, the trace indicates that member D9C1 was also running the same INSERT statement as D9C2, as shown in Example 11-37. Note that D9C1 also acquired the index tree P-lock during the execution of this insert statement. The index tree P-lock is used to serialize index page splits in a a data sharing environment. It is possible that the index page split was still in progress when the P-lock exit tried to obtain the page P-lock, and that our page was involved in the index page split. The trace was not detailed enough to allow us to determine whether this was the case.

*Example 11-37 INSERT statement issued by D9C2*

20:41:47.85846812 947637	1	61	INSERT	--> 'BLANK'	NETWORKID: USIBMSC	LUNAME: SCPD9C1	LUWSEQ: 1
24.61100724			START		LOCATION NAME	: DB9C	
					PKG COLLECTION ID	: RGDB07	
					PROGRAM NAME	: EMPADD	
					CONSISTENCY TOKEN	: X'18952F0501AACB52'	
					STATEMENT NUMBER	: 1	
					STATEMENT TYPE	: INSERT TYPE	
					CURSOR NAME	: N/P	
					QUERY COMMAND ID	: N/P	
					QUERY INSTANCE ID	: N/P	
					ISOLATION	: CS	
					REOPTIMIZATION	: NO	
...							
...							
20:41:47.85877139 947647	1	21	LOCK DETAIL	'BLANK'	NETWORKID: USIBMSC	LUNAME: SCPD9C1	LUWSEQ: 1
24.61119659							
-----							
LOCK RES TYPE: INDEX MANAGER TREE P-LOCK				DBID: 304	OBID: 16		
IRLM FUNC CODE : LOCK (NAME)				RETURN TOKEN: X'7F6AE150'	REQUEST TOKEN	: X'00000000'	
LOCK STATE : EXCLUSIVE				DB2 TOKEN : X'00A7F00021733428'	IRLM RETURN CODE	: 0	
LOCK ATTRIBUTES: MODIFY FORCE				PROP TO XES : YES	ASYN TO XES	: NO	
LOCK DURATION : INTEREST				REQUEST TYPE: P-LOCK	IRLM RETURN SUBCODE: B'0000000000000000'		
PARENT TOKEN : X'00000000'				GLOBAL/LOCAL: GLOBAL	OWNER	: 'BLANK'	
CACHED STATE : N/A					LOCK HASH VALUE	: X'00011530'	
QW0021CL: X'00'		QW0021U : X'010800001FED7F20'		QW0021CT: X'00000000'		QW0021FL: B'10010011'	
QW0021F3: B'10000000'		QW0021O : X'000000008B498C3F0'		QW0021IR: X'0000'		QW0021F2: B'00000000'	
-----							

This example demonstrates the wealth of information that is available in DB2 locking traces. We definitely do not recommend going down to this level of detail for each problem. Actually, the times you need to go down to this level of detail should be rare, but the information is available when you need it.



## Database and application design in data sharing

DB2 is designed to reduce the cost of data sharing enablement as much as possible. Various optimization techniques and algorithms have been implemented to enable you to take full advantage of your Parallel Sysplex environment. However, the data sharing enablement processing impact is sensitive to the system and application environment used. Several design and configuration options enable you to further reduce the cost.

Part 2, “Application concurrency and lock optimization” on page 63 of this publication has a detailed description about how locking and concurrency is affected by options you choose when designing your database and applications. Those guidelines also apply to DB2 data sharing environments.

In this chapter, we elaborate on some of the options that are specific and more critical in a data sharing environment to achieve best concurrency and performance.

The following topics are covered:

- ▶ Database design considerations
- ▶ Application design guidelines
- ▶ DB2 utilities in a data sharing environment

## 12.1 Database design considerations

It is important to do proper database design in a non-data sharing environment, but it is even more important to do so in a data sharing environment.

When running in a data sharing environment, there are mainly two types of processing impact that are specific to data sharing:

- ▶ Global locking
- ▶ Buffer pool coherency.

These result in additional activity that DB2 (and other z/OS components) have to do in order to be able to provide data integrity.

As the focus of this publication is concurrency and serialization, we mainly provide information about general database design guidelines for DB2 data sharing environments that help to reduce, or better control, the global locking processing impact.

### 12.1.1 Table space design

We now look at table space design parameters and definitions that can impact locking:

- ▶ Types of table space
- ▶ Lock size
- ▶ MAXROWS
- ▶ Page size
- ▶ Avoiding processing impact on system pages in your user objects
- ▶ APPEND YES
- ▶ CLOSE YES/NO

#### Types of table space

All types of table spaces are supported in a data sharing environment. The use of partitioned table spaces and universal table space (partition by range) in a DB2 data sharing environment can provide an advantage over other table space types. When using partitioned table spaces, table space locks are obtained at the partition level. Taking advantage of partition independence, you can reduce the processing impact for having inter-DB2 read/write interest or GBP-dependent partitions and having less child lock propagation. For example, partition one is being accessed by a DB2 utility on member 1, and a batch job is processing data from partition two. Even though rows from the same table are being accessed from different members, because the partitions are treated independently, each member acquires its own page set P-lock (in X state, for example) on separate partitions, and neither of the partitions will become group buffer pool dependent. Refer to “Selective partitions locking” on page 328 for more information.

Note that when you are using partitioned table spaces, more parent L-locks and more page set P-locks will be propagated to the coupling facility when a job accesses all partitions of a 4096 partition table space, compared to a non-partitioned table space.

Also note that in V9 you can no longer create simple table spaces. This makes partitioned table spaces the only table space type that allows the use of MEMBER CLUSTER. In V9, universal table spaces (UTS) do not allow MEMBER CLUSTER.

## Lock size

The usage of LOCKSIZE PAGE is a good design default for the table space lock size. Carefully consider the trade-off of using lock size ROW versus lock size PAGE. Row locking can have a considerable impact in a data sharing environment. The cost of row locking can be significantly higher than page locking if DB2 cannot use lock avoidance efficiently and you process a number of rows per page.

In a data sharing environment, page P-locks are used to provide the cross system page latch functionality when using row level locking. (Refer to 10.7, “Page P-locks” on page 329 for detailed descriptions about the mechanics of row level locking in a data sharing environment. Depending on the frequency of updates across the DB2 data sharing members, row level locking can produce a high rate of global (page P-lock) contention and can increase the CPU usage in a DB2 data sharing environment.

It might be better for the overall throughput if you use the MAXROWS 1 option in combination with LOCKSIZE PAGE, which is the logical equivalent of row level locking, but without the need for page P-locks on the data pages (as the L-locks on the page can now be used “protect” the data). Refer to 11.5.2, “LOCKSIZE ROW versus LOCKSIZE PAGE” on page 394 for a locking scenario using both options.

Note that MAXROWS *N* does not have to be MAXROWS 1. Simply reducing the number of rows per page can be sufficient to avoid the peak of the contention curve. Because of the impact on space usage, the usage of MAXROWS is usually only an option for relatively small tables.

## MAXROWS

Another use of MAXROWS *N* that applies to both LOCKSIZE PAGE and LOCKSIZE ROW is to reduce the time that is needed to find a page that has enough space in which to insert the row. This applies in particular to true variable length rows, or fixed length compressed rows. The free space indicator bits in the space map page could give a “false impression” that there is still enough room to hold an additional row, but after the page is visited, it turns out that there is not enough space after all.

This can result in many data pages being visited before a page with enough free space is found. If row level locking is in effect, this results in many page P-locks being requested. But even with page locking, it can result in many space map pages being visited (requiring a page P-lock) and (conditional) L-lock requests being issued for each of these pages into which DB2 tries to insert the row.

To reduce the space search effort, you can limit the number of rows by using MAXROWS *N*, where *N* is a value, where you are sure of that *N* rows will always fit on a page. This way, when the space map bits indicate there is free space, the row will fit when the page is visited. Once the page holds *N* rows, that page is marked full in the space map, and will not longer be considered for new inserts. This can drastically reduce the number of pages that have to be visited, and cut down on the number of page P-locks and L-locks that have to be requested.

But as always, this is a trade-off. As with MAXROWS 1, using MAXROWS *N* can waste space (on disk and in the buffer pool). Other maintenance challenges to consider are:

- ▶ What happens if you decide to compress the table space
- ▶ What happens if you no longer want to compress the table space
- ▶ What happens when a column needs to be added to the table?

You may have to adjust the value of *N* at that point, increasing or reducing its value appropriately. Otherwise, the problem that you are trying to solve may sneak back into the system through a “back door”.

## Page size

The general recommendation for the page size to have better concurrency is to use 4 KB pages.

However, in DB2 data sharing, there are situations where a larger page size can reduce the data sharing processing impact. The cost of data sharing can be reduced as the number of times that the coupling facility must be accessed is reduced. Particularly for sequential processing, larger page sizes can reduce this number. Fewer locks are taken on the larger page size, further reducing coupling facility interactions. On the other hand, large page sizes in combination with locksize row can reduce concurrency, as more rows will fit on a (larger) page.

Besides locking benefits, larger page sizes can help to reduce the number of getpages and improve disk space usage.

For table spaces that are always GBP-dependent in your installation, it can be worthwhile to go through the exercise of evaluating the trade-off for less concurrency versus a reduction in data sharing processing impact.

## Avoiding processing impact on system pages in your user objects

We have already seen that, in the case of row level locking, page P-locks are needed in addition to L locks to provide page consistency.

Page set P-locks are also used when updating space map pages (irrespective of the lock size parameter). To reduce page P-lock contention on space map pages, which is most often observed during heavy insert processing into GBP-dependent table spaces, you can use the MEMBER CLUSTER and TRACKMOD options on the CREATE TABLESPACE statement.

### MEMBER CLUSTER

In V9, the MEMBER CLUSTER option is only available for partitioned table space. It is not allowed for segmented or UTS table spaces. (Prior to DB2 9, you can also use it for simple table spaces, but those can no longer be created in V9.)

The member cluster option causes DB2 to manage space for inserts on a member basis instead of by using the clustering index, that is, each member inserts into its own set of pages. The use of MEMBER CLUSTER increases the number of space map pages (199 data pages per space map instead of 10K per space map for a 4 KB partitioned table space). Refer to “Space map page P-locks” on page 332 for more information.

The downside to using MEMBER CLUSTER is that data that is inserted by the SQL INSERT statement is not clustered by the implicit clustering index (the first index) or the explicit clustering index. In general, it tries to insert data in a place that is covered by the locally cached space map page on that member. If it cannot find space there, it continues to search through space map pages until it can find a place for which the space map page is available.

To reduce the processing impact of reacquiring page P-locks, a page P-lock is held longer for MEMBER CLUSTER table spaces. As each member will try to use its own set of pages to *insert* into, using MEMBER CLUSTER with LOCK SIZE ROW will also reduce page P-lock and page latch contention on *data* pages. Besides the impact of having to acquire page P-locks, row level locking without MEMBER CLUSTER, can result in excessive page P-lock contention on data pages if different members want to insert rows into the same page.

Although the use of MEMBER CLUSTER can dramatically improve insert performance, you also have to consider how the data will be accessed afterwards, as the data will no longer be inserted based on the clustering index. The use of MEMBER CLUSTER can have an adverse effect on processes that need to retrieve the data afterwards, especially if they used to access the data clustering index order.

### **TRACKMOD NO**

The TRACKMOD option specifies whether DB2 tracks modified pages in the space map pages of the table space or partition. When using the default (TRACKMOD YES), DB2 uses the space map page to track whether a page in a table space or partition has been modified. This information is used by incremental image copies to quickly determine which pages have been changed, and therefore need to be copied.

When an application is rapidly updating a table space from multiple members, contention can exist on the space map pages (page P-locks) because DB2 has to keep track of which pages have been changed in those space maps.

With TRACKMOD NO, DB2 does not track this change, and therefore avoids (the page P-lock) contention on the space map. However, when using incremental image copies with TRACKMOD NO objects, DB2 must use a table space scan to read all pages to determine whether the page has been changed and thus needs to be copied. Therefore, the use of TRACKMOD NO is not recommended if you depend on *fast* incremental image copies for your backup strategy.

If your application does heavy sequential inserts, consider also using the MEMBER CLUSTER option.

Refer to 11.5.3, “High INSERT data scenario” on page 398 for an example of the usage of MEMBER CLUSTER and TRACKMODNO.

### **APPEND YES**

DB2 9 for z/OS NFM also introduced the APPEND option. It commands to DB2 to ignore the clustering index during INSERT and LOAD operations. The syntax is as follows:

```
CREATE/ALTER TABLE ... APPEND YES
```

Using this option will reduce the amount spacemap page searching that is performed by DB2. DB2 will try to put the row at the end. However, although this may help INSERT performance, it may increase the need for table space reorganization, as query performance is likely to degrade, as the data is no longer in clustering order.

The behavior of APPEND, as far as space usage is concerned, is similar to the behavior of a table space that is defined as MEMBER CLUSTER and PCTFREE=FREEPAGE=0, sometimes called “pseudo-append”. The success depends on deletes and inserts being spread across DB2 members of data sharing group, as DB2 will switch between APPEND and INSERT mode.

Note that using the APPEND option may improve INSERT performance by cutting down on the time DB2 will spend to find the best place to put a row, by not using the clustering index, and not trying to minimize space usage. It can also lead to more space map page P-lock contention in a data sharing environment, as inserts will be mainly going to the end of the table space, making the last space map page a “hot” page. Therefore, in a data sharing environment when inserting from multiple members at the same time, you should define the table space as MEMBER CLUSTER in combination with APPEND YES. This way, each member will have its “own” space map page (and pages) in which to insert.

**Notes:**

- ▶ PTF UK41212 for PK65220 has extended the APPEND YES behavior to LOB table spaces.
- ▶ The PTF for PK81471 (open at the time of writing) will enhance the space search algorithm that is used by the APPEND logic. With this enhancement, APPEND YES will not try to reuse deleted space at the beginning of the object, but just extend. Therefore, the object is likely to grow faster than it otherwise would have, so you may need to REORG more frequently to reclaim space.

**CLOSE YES/NO**

Since APAR PQ69741, the CLOSE attribute is taken into account during pseudo-close processing, which means that DB2 will not close CLOSE NO page sets during pseudo-close processing, when attempting to remove group buffer pool dependency.

The use of CLOSE YES is a good design default. When objects are heavily used, they will remain open (and GBP-dependent) because of the activity against them. Infrequently used objects that use CLOSE YES are allowed to physically close the page set and drop out of group buffer pool dependency, thereby reducing the data sharing processing impact.

Using CLOSE NO gives you the ability to indicate to DB2 that these are objects that you want to keep physically open and GBP-dependent as long as possible. This can be because the transactions against these objects are very response time critical and you want to avoid the cost of physical open, or that the process of becoming group buffer pool dependent is very expensive for this object. For example, it always has a large number of pages in the local buffer pools of the different members, requiring many pages to be registered and potentially many locks that need to be propagated when the object becomes GBP-dependent again. For those objects, you may prefer a permanent low processing impact (of being GBP-dependent) compared to big spikes in activity when the object has to be physically opened or becomes GBP-dependent again. Refer to “Implications of using CLOSE NO” on page 336 for more information.

## 12.1.2 Index design

The locking related considerations for index design for DB2 non-data sharing that are described in 4.3, “Index design and locking considerations” on page 75 also apply to DB2 data sharing. In this section, we look at some index design concurrency considerations that are specific or more important for data sharing users:

- ▶ Non-partitioned secondary index
- ▶ Data partitioned secondary indexes
- ▶ Index page split (latch class 6)
- ▶ Random indexes
- ▶ Unused indexes
- ▶ Page size

**Non-partitioned secondary index**

Using partitioned table spaces in DB2 data sharing, and having different members accessing different partitions, can prevent GBP-dependency on the individual partitions, and reduce global locking and coupling facility processing impact.



However, this does not apply to non-partitioned indexes. In 6.4.2, “Utility operations with non-partitioned indexes” on page 175, we explain that DB2 can process a set of entries of a non-partitioned index (NPI) as a logical partition of the non-partitioned index. However, even when inserts, updates and deletes within each member only access a specific data partition, if index keys of the NPI are affected, the index space will become GBP-dependent. The effect of the global locking cost can be significant when using GBP-dependent, non-partitioned secondary indexes with utilities.

### **Data partitioned secondary indexes**

When you can use a data partitioned secondary indexes (DPSI) instead of a non-partitioned secondary index, you can prevent the DPSI from becoming GBP-dependent. A partition of a DPSI only contains keys that belong to the data partition it belongs to, much like the keys of the partitioning index. This way, if an application on member A deletes a row from data partition 1, and an application on member B deletes a row from partition 2, DB2 acquires an X page set P-lock on partition 1 and index partition 1 of the DPSI for member A, and an X page set P-lock on partition 2 and index partition 2 of the DPSI for member B. Both the data partitions and index partitions (of the DPSI) are not GBP-dependent. If the index had been an NPI, it would have become GBP-dependent.

We recommend using partitioned indexes within partitioned table spaces to take full advantage of partition independence.

However, the use of DPSIs may have an impact on query performance, so processes that have to work with data afterwards have to be taken into consideration as well when deciding whether or not to use a DPSI.

### **Index page split (latch class 6)**

Index page split occurs when an index page is full. By default, half of the index keys are kept on the old page, and half of the index keys are moved to a new page; the page is “split” into two pages, to allow for more index keys to be inserted into each of the pages.

Index page split is an expensive process, and particularly in a data sharing environment if the index is GBP-dependent.

For heavy insert workloads, page P-locks are acquired in indexes leaf pages. Also, latch class 6 is acquired for the index tree split that can cause contention in DB2 data sharing environments.

When an index page has to be split, an index tree P-lock is acquired (as well as a latch (showing up in LC 6)). The result is that the index cannot be used while the index split is in progress. DB2 does not want another process to only see half of the rows, or miss a number of keys, because they are in the process of being moved to another page.

In addition to the tree P-lock and latch, the index split process also results in two forced writes to the active log data sets in a data sharing environment. Therefore, it is vital that the DB2 logging environment is configured for optimal throughput, such as fast active log data sets, potentially striped, no paging on the log output buffer, and no or very small latch class 19 contention. However, a complete discussion about these topics is beyond the scope of this publication, as our focus is on locking and serialization.

The cost of index inserting and splitting should be an important consideration during the design phase of any high volume workload application.

An indication about the amount of leaf page split activity can be obtained, either from catalog table SYSIBM.SYSINDEXPART columns LEAFNEAR and LEAFFAR, and NLEAF from SYSIBM.SYSINDEXES, or from the real-time statistics table INDEXSPACESTATS columns REORGLAFTNEAR and REORGLAFTAFFAR. Information about the amount of index split contention is reported as latch class 6 contention in the DB2 statistics trace.

To reduce the amount of index page splits, the “normal” index design options are available. They are not really specific to data sharing, so we only touch on them briefly, but they need additional focus in a data sharing environment because of the extra processing impact involved.

### ***Free space***

Provide enough free space in the index pages (PCTFREE) to allow more keys to be added to existing pages, but also allow for free pages (FREEPAGE) to store new pages after an index page split.

When new index keys are inserted sequentially (ever increasing or decreasing), such as a time stamp, for example, you can use PCTFREE 0, as there will not be any new keys inserted into existing pages. However, the use of FREEPAGE can still be important, as there will be index page split activity, and the new page has to be stored somewhere. Therefore, it is important that “empty” pages are available to reduce the amount of time that is required to search for space to put the new page (while the index tree P-lock is held).

Note that sufficient free space not only helps reduce index page split, but also allows for efficient sequential read of the index, and for efficient sequential read of the data by way of the clustering index.

Monitor critical objects to ensure that appropriate free space is available and schedule REORGs when it drops below the threshold value.

### ***Asymmetric page split***

In DB2 9 new-function mode, the insert pattern of an index is monitored and analyzed. Based on the detected insert pattern, DB2 9 can split an index page by choosing from several algorithms (instead of the default 50/50). If a “nearly” ever-increasing or ever-decreasing sequential insert pattern is detected for an index, DB2 splits the index pages asymmetrically by using an approximately 90/10 split. This can significantly reduce the number of index page splits that are required.

Asymmetric split information is tracked in the actual pages that are being inserted into, so it is effective across multiple threads across different DB2 members. Make sure to install the PTF UK39357 for APAR PK62214. It enhances the tracking and detection logic, and it should enhance asymmetric split in a data sharing environment. It has an enhanced mechanism to detect “slightly out of order” keys. For example, DB2 can detect that inserted keys 3, 2, 1, 6, 5, 4, 9, 8, 7, 12, 11, and 10 represent an ascending pattern, and that an asymmetric split is appropriate here.

### ***Random indexes***

To set the stage from the beginning, random indexes can be a blessing or a curse. Index contention can be a major problem and a limiting factor for scalability of large applications, especially in a data sharing environment where a hot index page can be bounced back and forth between members (with index page P-lock contention, requiring that the page P-lock is negotiated).

in the past, a database administrator often had to resort to application assisted techniques, or adding additional columns to the index, to spread the keys. In DB2 9, NFM introduces random index keys, which can be set using the following command:

```
CREATE/ALTER INDEX ... column-name RANDOM  
instead of ASC or DESC
```

Using a randomized index key can dramatically reduce lock contention, as we demonstrated in 11.5.3, “High INSERT data scenario” on page 398.

However, a careful trade-off is required between lock contention relief and additional getpages, read/write I/Os, and an increased number of lock requests. Random indexes can be used with direct index matching predicates, but not for range predicates.

If the index is relatively small, or you have enough buffer pool resources to make the random index resident in the buffer pool, this solution can bring dramatic improvements, while eliminating much of the contention.

Using a randomized index key can also help reduce the number of page splits, especially when inserting ever increasing or decreasing values. After the key has been randomized, the inserts should be spread more evenly across all pages of the index. Provided there is enough free space available, this will reduce the number of page splits (and the LC6 contention that is the result of it).

## Unused indexes

Maintaining an index during insert, update, and delete operations or by way of DB2 utilities is not inexpensive, especially when many indexes exist on a table.

DB2 9 for z/OS added the LASTUSED column to the SYSIBM.SYSINDEXSPACESTATS RTS table. It indicates the day that the index was last used by DB2. The value is updated once every day (the first time the RTS statistics are externalized after midnight).

An index is considered used when it is used:

- ▶ By an access path for a query or during a FETCH operation
- ▶ By a searched UPDATE / DELETE SQL statement
- ▶ As a primary index for referential integrity checking
- ▶ To support foreign key access

This capability allows you to revisit your indexes. With RTS detecting index usage at run time, you can determine that an index is no longer used, and, after verifying the access path, it is a candidate to be dropped.

## Page size

The larger index page sizes that were introduced in DB2 9 for z/OS can also help reduce the index leaf page splits, which, as we discussed before, are especially painful for GBP-dependent indexes in a data sharing environment.

When more index keys can fit on a page, it will take longer to fill the page, and thereby reduce the number of index page splits and the number of index tree P-lock contentions.

However, as is the case for larger data page sizes, using larger index pages also gives you the ability to aggravate the index buffer pool hit ratio for random index access.

## 12.2 Application design guidelines

In this section, we focus on application design guidelines for achieving the best concurrency control in a DB2 data sharing environment.

### 12.2.1 Maximize lock avoidance

The path length of acquiring a lock from IRLM compared to acquiring a latch is considerably higher. If the lock has to be propagated to the CF, it is even more expensive. In addition, every time you acquire a lock, you could experience contention on the object, for example, due to the granularity of the lock modes of z/OS XES. Resolving this contention can be a lengthy and expensive process.

Therefore, every lock you can avoid improves the overall resource consumption in a data sharing environment. The impact of lock avoidance is therefore more important than in a non-data sharing environment.

The number of child locks that need to be propagated to the CF affects the cost of global locking. Processing a large number of qualifying rows could cause a considerable processing impact. Lock avoidance reduces the impact of child lock propagation by reducing the number of S locks. Lock avoidance, if successful, eliminates calls to the IRLM to request a page lock or row lock.

The efficiency of lock avoidance depends on three parameters:

- ▶ The CURRENTDATA value
- ▶ The ISOLATION level of your plan and package
- ▶ The frequency of commits

#### CURRENTDATA and ISOLATION

RR isolation does not allow lock avoidance, because it always makes sure that the data remains unchanged, where lock avoidance allows flexibility of change even if you are positioned on the row. RS isolation allows the use of lock avoidance for non-qualifying rows.

The CURRENTDATA bind option enables you to determine which rows can be subject to lock avoidance when using CS isolation. Specifying a value of YES indicates to DB2 that you want to keep the contents of the row on which you are positioned the same. This is implemented through the use of an S lock on the row or page. Using CURRENTDATA(YES) effectively disables lock avoidance for qualifying rows.

You must try to use CURRENTDATA(NO) and ISOLATION(CS) as much as possible to maximize the effect of lock avoidance.

#### Global commit log sequence number

For objects without inter-DB2 R/W interest, lock avoidance depends on the highest committed *log record sequence number* at the page set level within the DB2 member. For objects with inter-DB2 R/W interest, lock avoidance depends on the *highest committed log record sequence number* for the entire data sharing group. A recent commit increases the efficiency of lock avoidance. It is, therefore, important that all your applications issue commits at frequent intervals to ensure that the lock avoidance efficiency of the GBP-dependent page sets does not deteriorates.

## Commit frequency

It is a long standing recommendation to take intermediate commit points for long running application processes to reduce the amount of backout processing and to reduce the amount of forward processing on the subsequent restart following a failure. This recommendation needs to be re-enforced in data sharing environments. You should make frequent intermediate commit points to ensure efficient global avoidance for all application processes and to free up IRLM, MVS XES, and CF resources.

DB2 provides warning messages for long running units of recovery when user-established DSNZPARM thresholds are exceeded:

- ▶ DSNR035I for URCHKTH, for the number of checkpoint cycles
- ▶ DSNJ031I for URLGWTH, for the number of log records written
- ▶ LRDRTHLD for a long running reader

Refer to 8.3.4, “Periodic monitoring of DB2 system level concurrency related IFCIDs” on page 243 for details about this topic.

Your DB2 data sharing installation should have a very aggressive procedure to monitor the long running uncommitted unit of recovery, which leads to very effective lock avoidance.

Refer to 2.3.1, “Lock avoidance control” on page 36 and 10.12, “Lock avoidance in data sharing” on page 362 for additional information about lock avoidance.

## 12.2.2 Table space lock duration

Before the advent of locking protocol 2, using RELEASE(DEALLOCATE) was often recommended to reduce XES contention (IX and IS table space locks leading to X and S XES locks). Since locking protocol 2, the new mapping of parent L-lock “intent to exclusive IX” to the XES S “Share” lock has virtually eliminated the need for using the RELEASE(DEALLOCATE) bind option to reduce XES contention.

There are other reasons for using RELEASE(DEALLOCATE), such as:

- ▶ The effective benefits of thread reuse with RELEASE(DEALLOCATE) for small, and very frequently executed, plans and packages, or
- ▶ The use of RELEASE(DEALLOCATE) in batch programs that issue many commits, but want to avoid the reset at commit for sequential detection, index lookaside, and IPROC.

However, you should also consider the “downsides” of using RELEASE(DEALLOCATE). In many installations, they often outweigh the benefits. The disadvantages of using RELEASE(DEALLOCATE) are the virtual storage capacity, the additional EDM pool storage consumption, and potential resource contention when performing concurrent rebind and DDL. This is especially problematic in a CICS environment with heavy thread reuse when using a single plan.

For more information about lock durations, refer to 5.2, “Optimizing concurrency and locking” on page 100, and for more information about the RELEASE bind option, refer to section 11.1.3, “RELEASE:”, in *DB2 UDB for z/OS: Design Guidelines for High Performance and Availability*, SG24-7134.

The recommendation for the ACQUIRE bind option is to use ACQUIRE(USE) for better concurrency.

Refer to 5.13.1, “Protected entry threads” on page 143 for more considerations about ACQUIRE.

### 12.2.3 P-locks and isolation levels

Page set or partition P-lock are acquired regardless of the isolation level for the application. This applies to both read-only and write applications. The page set or partition P-locks are always propagated to the coupling facility.

Example 12-1 shows the DISPLAY..LOCKS output for two reader applications, one on each member. The SQL issued from both D9C1 and D9C2 members is:

```
SELECT .. FROM glwtprj WHERE proj_no=hv WITH UR
```

*Example 12-1 Isolation UR reader example*

---

```
DSNT360I  -D9C1 *****
DSNT361I  -D9C1 *   DISPLAY DATABASE SUMMARY
              *   GLOBAL LOCKS
DSNT360I  -D9C1 *****
DSNT362I  -D9C1      DATABASE = RITA1229 STATUS = RW
              DBD LENGTH = 32294
DSNT397I  -D9C1
NAME      TYPE PART  STATUS          CONNID  CORRID      LOCKINFO
-----
GLWSPRJ   TS        RW                      H-S,PP,I
-         MEMBER NAME D9C2
GLWSPRJ   TS        RW                      H-S,PP,I
-         MEMBER NAME D9C1
*****  DISPLAY OF DATABASE RITA1229 ENDED  *****
DSN9022I  -D9C1 DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

---

Note that even for UR readers, page set P-locks are acquired (and propagated).

Page P-locks, on the other hand, are *only* used when there is inter-DB2 write/write interest on a page set. Page P-locks are acquired in “X” exclusive state. This applies to any isolation level for applications that perform update (I/U/D) operations. Only in special cases do readers acquire “S” share page P-lock, including the following items:

- ▶ Repeatable Read data and index scan with row level locking
- ▶ Referential Integrity checking

A reader process using a UR, CS, or RS isolation level does not acquire any P-locks on index or data pages.

### 12.2.4 Use uncommitted read isolation

UR isolation has the lowest locking cost, because it does not acquire parent or child L-locks. It only acquires a mass delete lock. If your application can manage the potential inconsistency or if your data is kept consistent, you should exploit UR isolation to reduce the cost of locking. UR isolation is available at the package/plan level and also at the statement level.

### 12.2.5 Sequences and identity columns

Some SQL features have a special behavior in a DB2 data sharing environment that you should be aware of when designing data sharing enabled applications. Sequences and identity columns are such features. In this section, we use sequences as a way to illustrate these considerations, but they also apply to identity columns.

The CACHE option of the CREATE SEQUENCE statement is a performance and tuning option that directs DB2 to preallocate a specified number of sequential values in memory. Every time a cache is allocated, the SYSIBM.SYSSEQUENCES table is updated and a forced log record is written for the update. Using ORDER or NO CACHE will result in a SYSIBM.SYSSEQUENCES table update and the associated forced log write every time a new value is generated by DB2.

Specifying CACHE provides faster access to the sequence, because values are assigned from the cache, which reduces the number of SYSIBM.SYSSEQUENCES updates and the associated forced log records. The SYSIBM.SYSSEQ table space is defined with the MAXROW 1 attribute, which removes any contention among sequences.

In a data sharing environment, if ORDER is specified, then NO CACHE is implicit even if CACHE *n* is specified. In a non-data sharing environment, the numbers are always assigned in strict numeric order even if NO ORDER was specified. Thus, specifying NO CACHE is not a good idea. There are opposing considerations when deciding on the CACHE (or ORDER) option:

- ▶ Using a CACHE > 0 can result in generating values for the sequence across multiple DB2 members that may not be in strict numeric order.
- ▶ Using a NO CACHE or ORDER can result in excessive log force writes and cross invalidation of the SYSIBM.SYSSEQUENCES table pages.

### Using CACHE > 0

DB2 always generates sequence numbers in order of request. However, when a sequence is shared across multiple members of a data sharing group, each DB2 member allocates its own cache of unique consecutive numbers for the sequence. Therefore, in situations where transactions from different members are requesting the next sequence number from the same sequence, values assigned for the sequence across multiple DB2 members may not be in strict numeric order.

For example, assume that members DB2A and DB2B share a sequence named SEQ1 that starts with 1, increments by 1, and has cache 20. If the transaction that is associated with DB2A makes the first request for a sequence number, DB2A allocates a cache of 20 values (from 1 to 20) and the value of 1 is provided to the application. If the transaction that is associated with DB2B makes the next request for a sequence number, DB2B allocates its own cache of 20 values (from 21 to 40) and the value of 21 is provided to the application. Assuming that sequence number requests continue to arrive from the transactions that are associated with members DB2A and DB2B in this manner (one from DB2A and then one from DB2B), the values assigned for that sequence are 1, 21, 2, 22, 3, 23, and so on. Although the numbers are in sequence within each DB2, the numbers assigned across multiple DB2 members are not in strict numeric sequence.

### Using NO CACHE

The NO CACHE option will avoid the creation of sequence numbers or identity column values out of sequence. However, this full sequencing will come at a premium cost. Without data sharing, the recoverability of the sequence/identity column is provided by the log write ahead protocol.

With data sharing, we need to consider the recoverability and the inter-system consistency. DB2 uses log write force each time the entry in SYSIBM.SYSSEQUENCES is updated. This can lead to an excessive amount of force log write I/Os, which in turn reduces the log write bandwidth dramatically. What is the throughput of an ONLINE LOAD or an INSERT INTO SELECT using a sequence or identity column? Their insert rate is limited by the number of I/Os to the active log that can be performed per second.

**Tip:** Avoid using a small CACHE size in data sharing if your application can tolerate rows out of sequence if rows are inserted in more than one member, or if sequence values are obtained on more than one member.

If you only need to obtain a unique number, not in any particular order, without the need for a particular ordering, consider using the `GENERATE_UNIQUE()` built-in function. It generates a unique number and does not require any catalog, logging, or CF activity.

The extra logging is performed, even if you are in single member data sharing.

## 12.2.6 Create member affinity

Creating member affinity is a complex process that requires intervention in job scheduling, DDL, and application design. Creating member affinity is beneficial only if it enables you to reduce the number of GBP-dependent page sets. The technique is not relevant in a read-only environment because you need at least one updating member to become GBP-dependent. If you fail to control all of the parameters, you might end up with reduced availability without the performance benefits.

### Job scheduling

You have two basic options for creating member affinity:

- ▶ Isolating a set of tables by routing all work for a set of specific tables to one member
- ▶ Isolating the work to a partition

#### *Isolating a set of tables*

Isolating the work to a specific table is an option, if you can isolate sets of transactions and batch programs that use a subset of your tables. If your accounting application has its own set of tables, that work can be routed to a single member. It is possible to restrict access to data to a single member. For example, if you want access to a table space named NOSHARE limited only to DB2A, you could assign NOSHARE to a previously unused buffer pool, such as BP25, using the `ALTER TABLESPACE` command. Do not define a GBP that corresponds to BP25, and assign BP25 a size of zero on any other DB2 in the group. This prevents the other members of the group from attempting to use this buffer pool, and therefore from accessing table space NOSHARE. This technique is applicable only if the number and size of tables that you want to group is sufficiently large. If not, you probably do not want to dedicate a buffer pool because that would increase your demand for virtual storage without any real benefit. The decrease in virtual storage efficiency is probably more important than the data sharing processing impact of the occasional access to the data from another member.

Another consideration is the impact on your error handling routines and availability. A job could be executed on the wrong member, either planned or as the result of an operational error. Your contingency scenario should be prepared to intervene on your buffer pool configuration in case you want to reroute the work of other members.

#### *Isolating the work to a partition (partition level)*

If you cannot restrict the use of a table to a member, it might still be possible to isolate the activity to a partition. For batch jobs, it can be quite easy to determine the key range involved in the execution. If a batch program is split over multiple members for performance reasons, you need to partition in such a way that your partition ranges match the boundaries of your jobs. The requirement to use different page sets for each of the members cannot be applied to any non-partitioning index, because there is only one page set for the entire table space.



Determining the range in a transaction environment can be much more complicated or even impossible, because the range might depend on the processing logic. You could be halfway through the transaction before you know which partition is used.

Having one transaction out of 1,000 crossing the line and updating a row in a partition assigned to another member might in fact be worse than a full data sharing solution with random scheduling of the transactions. Maintaining the GBP-dependency has a cost, but going in and out of this state is probably worse. If an update is performed to the partition of the other member, you have a R/W interest by both members on the same page set. Because you now have inter-DB2 R/W interest and the page set has become GBP-dependent, DB2 needs to register in the CF all pages of the page set that reside in the local buffer pool. Provided that the update was a one off, remember that if only one transaction in 10,000 crosses the line, DB2 will declare the page set non-GBP-dependent some time after the members that are not supposed to update the partition perform the R/O switch, assuming the page set is defined as CLOSE YES. DB2 physically closes the page set after it has been R/O for a certain amount of time if there is one R/W member and the R/O members do not use the page set. The worst scenario is the second transaction again crossing the line after you have made the R/O switch, because the entire process is repeated.

### Potential reduced availability

Not only can member affinity cause problems with the system because of “rogue” updates that make objects become GBP-dependent when the whole design was to avoid GBP-dependency, it can also affect availability.

If all transactions are well-behaved and the job scheduler does a perfect job routing different work to different members, the result will be that each member will have an X-mode page set P-lock on each of the objects it is accessing, as each member will be the only updater of the object.

In case a member abnormally terminates, each X-mode page set P-lock that it owns is converted to a retained X-mode page set P-lock. This prevents access from any of the other members until the failed member has completed the forward recovery phase of its restart. So instead of only the pages that this member was actively working on (retained L-locks) being made unavailable to the other members, the entire page set is unavailable until the abnormally terminated member has been restarted.

**Note:** The reduction in data sharing processing impact that can be achieved by implementing affinity routing techniques has to be evaluated against the loss of availability in case a member abnormally terminates.

## 12.2.7 IMMEDIATEWRITE

Consider the situation where one transaction updates DB2 data using INSERT, UPDATE, or DELETE, and then, before completing phase 1 of commit, it spawns a second transaction that is dependent on the updates that were made by the first transaction. This type of relationship is referred to as “*ordered dependency*” between transactions.

Now consider the following scenario: We have a 2-way data sharing group DB9CG with members D9C1 and D9C2. Transaction T1, running on member D9C1, makes an update to a page. Transaction T2, spawned by T1 and dependent on the updates made by T1, runs on member D9C2. If transaction T2 is not bound with isolation repeatable read (RR), and the updated page (on D9C1) has been used previously by D9C2 and is still in its local buffer pool, there is a chance, due to lock avoidance, that T2 uses an old copy of the same page in the virtual buffer pool of D9C2 if T1 still has not committed the update.

Here are some possible workarounds for this problem:

- ▶ Execute the two transactions on the same member.
- ▶ Bind transaction T2 with ISOLATION(RR).
- ▶ Do not use ambiguous cursors (T2 does not use them).
- ▶ Make T1 commit before spawning T2.
- ▶ Use IMMEDIATEWRITE.

The *immediate write* option allows the user to specify when DB2 should write updated GBP-dependent buffers to the Coupling Facility. You can either wait until commit or rollback, or you can have DB2 write the changed page as soon as it has been updated (without waiting on the commit). The IMMEDIATEWRITE parameter can either be specified at BIND time or by using the IMMEDIATEWRI DSNZPARM parameter.

The default processing (NO) changed pages are written (at the latest) during phase 1 of commit processing. When you use IMMEDIATEWRITE YES, changed pages are written to the group buffer pool as soon as the buffer updates are complete (so definitely before committing). Remember that before writing pages to the GBP, DB2 makes certain that the updates are first written to the active log data set on disk. Specifying this option may impact performance.

You have to be especially careful when setting IMMEDIATEWRI DSNZPARM to YES. Doing so overrides the IMMEDIATEWRITE bind at the plan and package level.

Most ordered dependent transactions are spawned as part of the commit processing. As the GBP writes are done as part of phase 1 of commit processing, using IMMEDIATEWRITE(NO) for those transactions works.

If you have identified transactions that spawn an ordered dependent transaction before commit, we recommend that you use the IMMEDIATEWRITE(YES) BIND option for the spawning transaction and not use the DSNZPARM option because of its impact on all plans and not just the one that is causing the problem.

**Tip:** Avoid using the IMMEDIATEWRI DSNZPARM and use the IMMEDIATEWRITE BIND option only when required.

## 12.3 DB2 utilities in a data sharing environment

Utilities behave pretty much the same when running in a data sharing environment compared to a non-data sharing environment.

Like regular transactions that access an object, utilities also have to acquire page set P-locks. For example, if a transaction is updating a table space on one member and is the only one accessing that object, running a RUNSTATS utility against that table space on another member will make the table space GBP-dependent.

### 12.3.1 Utilities and partitioned objects

When you use partitioned table spaces, you have the ability to run multiple jobs in parallel, each job dealing with one or a number of partitions of the same table space (or index). This is called *partition independence*. This also means that you can spread these jobs across more than one member of a DB2 data sharing group; one job reorganizing partition 1 on member 1, and another job doing the same for partition 2 on member 2. Although this is possible, you have to be aware of the effects of running multiple jobs on multiple members against the same table space, especially when non-partitioned indexes are involved.

For example, when you have a very large partitioned table space with a partitioning (partitioned) index, a data partitioned secondary index (DPSI), and a non-partitioning index (NPI), you can run a job against partition 1-20 on member A and another job against partition 21-40 on member B. This situation lets you use the CPU power of both members in parallel to speed up the REORG process.

Using partition independence, each job can process its own range of partitions without blocking the other job that is processing a different range of partitions. This works perfect for the data partitions and index partitions of the partitioned indexes (partitioning and DPSI). However, the NPI is a single indexset (with logical partitions). When the jobs, each one running on a different member, are accessing the same NPI with an intent to update, the NPI becomes group buffer pool dependent. This is, for example, the case for a REORG SHRLEVEL NONE during index maintenance of the NPI. When the NPI is GBP-dependent, the index pages have to be written to the group buffer pool, and index page P-locks have to be acquired.

As utilities usually process a large number of pages, even though DB2 tries to throttle the work to the CF a bit, index maintenance driven from multiple members can put a great deal of pressure on the system, where the lock structure and group buffer pool can get flooded with requests, and the index page P-locks can cause a considerable amount of contention.

This mostly applies to utility processes that can maintain non-partitioning indexes from multiple members, such as:

- ▶ Multiple REORG SHRLEVEL NONE jobs running on multiple members against multiple partitions of the same table space during index maintenance where one or more NPI exist.
- ▶ In V8, multiple REORG SHRLEVEL REFERENCE or CHANGE jobs on multiple members against multiple partitions of the same table space where one or more NPI exist, during the BUILD2 phase.

Note that this no longer applies to DB2 9, as the BUILD2 phase has been eliminated. As a consequence of the way this has been implemented, you can no longer run multiple REORG jobs against the same partitioned table space at the same time (on the same member or from different members).

- ▶ Multiple LOAD jobs, loading different partitions from the same table space in parallel from multiple members.

If the partitioned table space only had partitioned indexes (partitioning or secondary), running multiple jobs against different partitions on different members will not trigger any of them to become GBP-dependent.

**Tip:**

- ▶ In a data sharing environment, we recommend spreading DB2 utility jobs for different objects across the data sharing group. Run all jobs against table space A on member 1, and all jobs against table space B on member 2.
- ▶ Keep concurrent DB2 utilities for the same object running in the same member to minimize lock contention and group buffer pool access.

### 12.3.2 Utilities and pageset P-locks

Drains and claims were introduced in DB2 to serialize access between SQL, utilities, and DB2 commands. They are also used in data sharing environments. A drain on a page set or partition is a global logical lock. Drain logical locks always propagate to the coupling facility. This is true even for non-GBP-dependent page sets or partitions. In data sharing, claim counters for that page set or partition are maintained locally in each member. The first claimer obtains a share S-lock on the page set or partition and the counter is incremented whenever other claims occur.

In 3.5, “Claims and drains” on page 53 and 6.1, “Claims and drains for concurrency control” on page 162, we explain in detail how DB2 utilities and commands can obtain access to DB2 objects through claims (in the case of COPY SHRLEVEL(CHANGE) and RUNSTATS SHRLEVEL(CHANGE), drains, and their own set of compatibility rules. The same rules apply to DB2 data sharing, and in addition, there is a global locking cost associated when a page set or partition is GBP-dependent. Here we use some examples to describe the role of db2 data sharing global locking. The following scenarios are presented:

- ▶ Concurrency of utilities and SQL - Inter-DB2 read/write
- ▶ Concurrency of utilities and SQL - Inter-DB2 write/write
- ▶ Utilities compatibility rules

#### **Concurrency of utilities and SQL: inter-DB2 read/write**

In this sample scenario, member D9C1 executed a COPY SHRLEVEL(REFERENCE) utility and, in member D9C2, a transaction tx1 issued the following SQL statement:

```
SELECT emp_no, proj_no, act_no, emptime  
FROM RITA1229.GLWTEPA  
WHERE emp_no=5003 WITH CS;
```

Example 12-2 on page 435 shows in DISPLAY...LOCKS ONLY that DB2 member D9C1 has “SIX” share intent to exclusive mode and member D9C2 has “IS” intent to share mode page set or partition P-lock in table space GLWSEPA.

*Example 12-2 Utilities and SQL in data sharing: inter-DB2 read/write*

```

DSNT360I  -D9C2 *****
DSNT361I  -D9C2 *   DISPLAY DATABASE SUMMARY
                *   GLOBAL LOCKS
DSNT360I  -D9C2 *****
DSNT362I  -D9C2      DATABASE = RITA1229 STATUS = RW
                DBD LENGTH = 28256
DSNT397I  -D9C2
NAME      TYPE PART  STATUS              CONNID  CORRID      LOCKINFO
-----
GLWSEPA  TS          RW,UTRO              UTILITY  DB2R2RE3    H-IX,W,A
-
-          AGENT TOKEN 189
-          MEMBER NAME D9C1
GLWSEPA  TS          RW,UTRO              H-SIX,PP,I
-          MEMBER NAME D9C1      (C0)
GLWSEPA  TS          RW,UTRO              H-IS,PP,I
-          MEMBER NAME D9C2
GLWSEPA  TS          RW,UTRO              TS0      DB2R2      H-IS,S,C
-          AGENT TOKEN 6
-          MEMBER NAME D9C2
29        TB          TS0      DB2R2      H-IS,T,C
-          AGENT TOKEN 6
-          MEMBER NAME D9C2
GLWXEPA1 IX          RW                  H-S,PP,I
-          MEMBER NAME D9C2
GLWXEPA3 IX          RW                  H-S,PP,I
-          MEMBER NAME D9C2
GLWXPJA1 IX          RW                  H-S,PP,I
-          MEMBER NAME D9C2
***** DISPLAY OF DATABASE RITA1229  ENDED *****
DSN9022I  -D9C2 DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION

```

In member D9C1, the utility COPY SHRLEVEL(REFERENCE) obtains a write drain lock with duration 'allocation' (H-IX,W,A), while the SQL statement issued in member D9C2 for the same table space has a cursor stability claim class.

The utility COPY SHRLEVEL(REFERENCE) updates the change bits in the space map page for the table space, so there is an inter-DB2 read/write interest and the page set or partition is GBP-dependent.

Table 12-1 shows the claim and drain classes obtained in this scenario.

*Table 12-1 Table space GLWSEPA claim and drain classes*

Member	Write drain	CS Read drain	Table space lock
D9C1	1	0	None
D9C2	0	1	IS

## Concurrency of utilities and SQL: inter-DB2 write/write interest

For the inter-DB2 write/write scenario, member D9C1 executes a REORG SHRLEVEL(CHANGE) utility on PART 1 and a transaction tx2 on member D9C2 that issues the following SQL statement:

```
UPDATE RITA1229.GLWTEMP
SET LASTNAME='G', FIRSTNME='RITA',WORKDEPT=1
WHERE EMP_NO = 1
WITH CS;
```

The -DISPLAY...LOCKS ONLY command was executed when the REORG SHRLEVEL(CHANGE) utility entered the LOG phase and tx2 transaction still had not committed. We used the default DRAIN WRITERS option for the REORG SHRLEVEL(CHANGE) indicating the type of drain that has to be acquired during the log processing phase. The -DISPLAY UTILITY command output at that time is as follows:

```
DSNU105I  -D9C2 DSNUGDIS - USERID = DB2R2
          MEMBER = D9C1
          UTILID = REORG
          PROCESSING UTILITY STATEMENT 1
          UTILITY = REORG
          PHASE = LOG    COUNT = 0
          NUMBER OF OBJECTS IN LIST = 1
          LAST OBJECT STARTED = 1
          STATUS = ACTIVE
```

Example 12-3 shows the DISPLAY...LOCKS ONLY output.

*Example 12-3 utilities and SQL in data sharing: inter-DB2 write/write*

---

```
DSNT360I  -D9C2 *****
DSNT361I  -D9C2 *   DISPLAY DATABASE SUMMARY
          *   GLOBAL LOCKS
DSNT360I  -D9C2 *****
DSNT362I  -D9C2      DATABASE = RITA1229  STATUS = RW
          DBD LENGTH = 28256
DSNT397I  -D9C2
NAME      TYPE PART  STATUS                      CONNID  CORRID      LOCKINFO
-----
GLWSEMP   TS      0001 RW,UTRW                      UTILITY  DB2R2RE2    H-IX,W,A
-                                     AGENT TOKEN 274
-                                     MEMBER NAME D9C1
GLWSEMP   TS      0001 RW,UTRW                      H-IX,PP,I
-                                     MEMBER NAME D9C1
GLWSEMP   TS      0001 RW,UTRW                      H-IX,PP,I
-                                     MEMBER NAME D9C2      (C0)
GLWSEMP   TS      0001 RW,UTRW                      TSO      DB2R2      H-IX,P,C
-                                     AGENT TOKEN 387
-                                     MEMBER NAME D9C2
GLWXDPT1  IX                      RW                      H-S,PP,I
-                                     MEMBER NAME D9C2
GLWXEMP1  IX      L*   RW,UTRW                      H-IX,PP,I
-                                     MEMBER NAME D9C1
GLWXEMP1  IX      L*   RW,UTRW                      H-IX,PP,I
-                                     MEMBER NAME D9C2      (C0)
GLWXEMP2  IX      L*   RW,UTRW                      H-IX,PP,I
-                                     MEMBER NAME D9C1
```

```

GLWXEMP2 IX      L*   RW,UTRW                                H-IX,PP,I
-               MEMBER NAME D9C2              (C0)
GLWXEMP3 IX      D0001 RW,UTRW                                H-S,PP,I
-               MEMBER NAME D9C2
***** DISPLAY OF DATABASE RITA1229   ENDED *****
DSN9022I  -D9C2 DSNRDDIS 'DISPLAY DATABASE' NORMAL COMPLETION

```

---

You can see that both members has a “IX” intent to exclusive page set P-lock (in this scenario, we used a partitioned table space). The REORG utility on member D9C1 has obtained a write drain lock (H-IX,W,A) and is then waiting for the claim class count to go to zero. The LOCKINFO shows H-IX,P,C for transaction Tx2.

Note that in the example, index GLWXEMP3 is a partitioned index while GLWXEMP1 and GLWXEMP2 are secondary non-partitioned indexes.

We also executed the DISPLAY BUFFERPOOL...GBPDEP(YES) command for BP2 and BP3, which are table space and index buffer pools, respectively, using this command:

```
-DIS BUFFERPOOL(BPx) DBNAME(RITA1229) SPACENAM(*) GBPDEP(YES)
```

The output in Example 12-4 indicates that GLWSEMP table space partition 01 and indexes GLWXEMP1 and GLWXEMP2 are GBP-dependent.

#### Example 12-4 DISPLAY BUFFERPOOL...GBPDEP(YES) output

---

```

DSNB401I  -D9C2 BUFFERPOOL NAME BP2, BUFFERPOOL ID 2, USE COUNT 3
...
...
-----PAGE SET/PARTITION LIST INFORMATION-----
-----DATA SHARING INFO-----
          TS GBP  MEMBER  CASTOUT  USE  P-LOCK
DATABASE SPACE NAME  INST PART IX DEP   NAME    OWNER  COUNT STATE
=====
RITA1229  GLWSEMP      0001 0001 TS  Y  D9C2      Y      1  IX
                                   D9C1      1  IX
DSN9022I  -D9C2 DSNB1CMD '-DIS BUFFERPOOL' NORMAL COMPLETION DSN9022I  -D9C2 DSNB1CMD '-DIS
BUFFERPOOL' NORMAL COMPLETION
*****
DSNB401I  -D9C2 BUFFERPOOL NAME BP3, BUFFERPOOL ID 3, USE COUNT 27
...
...
-----PAGE SET/PARTITION LIST INFORMATION-----
-----DATA SHARING INFO-----
          TS GBP  MEMBER  CASTOUT  USE  P-LOCK
DATABASE SPACE NAME  INST PART IX DEP   NAME    OWNER  COUNT STATE
=====
RITA1229  GLWXEMP2      0001      IX  Y  D9C2      Y      1  IX
                                   D9C1      1  IX
          GLWXEMP1      0001      IX  Y  D9C2      Y      1  IX
                                   D9C1      1  IX
DSN9022I  -D9C2 DSNB1CMD '-DIS BUFFERPOOL' NORMAL COMPLETION

```

---

Table 12-2 shows the claim and drain classes during the REORG LOG phase.

Table 12-2 Claim and drain classes in REORG log phase

Member	Write drain	CS Read drain	Table space partition 1 lock
D9C1	1	0	None
D9C2	1	0	IX

In this scenario, as both DB2 members have “IX” intent to share page set P-lock, S, U, and X mode child locks will be propagated. Refer to Table 10-7 on page 326 for more information. In our case, the tx2 transaction was accessing partition 01 where the utility was running.

## Utilities compatibility rules

To illustrate the utilities compatibility rules in a data sharing environment, we run two utilities. Each utility runs on a different member against the GLWSEMP partitioned table space. We ran a COPY SHRLEVEL(CHANGE) on member D9C1, and a RUNSTATS SHRLEVEL(CHANGE) on member D9C2. Example 12-5 shows the partition P-locks for both DB2 members.

Example 12-5 Page set P-locks for concurrent utility access

```

DSNT360I  -D9C1 *****
DSNT361I  -D9C1 *   DISPLAY DATABASE SUMMARY
              *   GLOBAL LOCKS
DSNT360I  -D9C1 *****
DSNT362I  -D9C1      DATABASE = RGDDB10  STATUS = RW
              DBD LENGTH = 28256
DSNT397I  -D9C1
NAME      TYPE PART  STATUS              CONNID  CORRID      LOCKINFO
-----
GLWSEMP   TS      0001 RW,UTRW              H-IS,PP,I
-          MEMBER NAME D9C2
GLWSEMP   TS      0001 RW,UTRW              H-SIX,PP,I
-          MEMBER NAME D9C1      (CO)
GLWSEMP   TS      0002 RW,UTRW              H-IS,PP,I
-          MEMBER NAME D9C2
GLWSEMP   TS      0002 RW,UTRW              H-SIX,PP,I
-          MEMBER NAME D9C1      (CO)
...

```

For details about utilities’ compatibility, refer to each utility’s description in *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.





# Part 5

## Appendixes

This part only contains Appendix A, “System topology and workload” on page 441.





# A

## System topology and workload

This appendix describes the environment setup for this project.

The following topics are covered:

- ▶ “Hardware and software set up” on page 442 describes the components of our test environment.
- ▶ “Stored procedures workload” on page 442 describes the database model of the workload used for our tests.

## Hardware and software set up

For our hardware, we have a sysplex complex in an LPAR using a 2094 IBM System z9® with two shared CPs and 4 GB of real storage. The OS level is z/OS V1R10. The DB2 9 for z/OS was updated to level D08100 at the start of our project. We added more recent maintenance levels.

This section describes the environment on which we perform our tests. Figure A-1 shows our starting configuration. We use three DB2 9 for z/OS subsystems. DB9A is a stand-alone subsystem in LPAR SC63, while D9C1 and D9C2 are the two data sharing members on LPARs SC63 and SC64, respectively. The processors in WTSCPLX2 are shared across all three LPARs except SC64, which does not have the zIIP and zAAP processors defined.

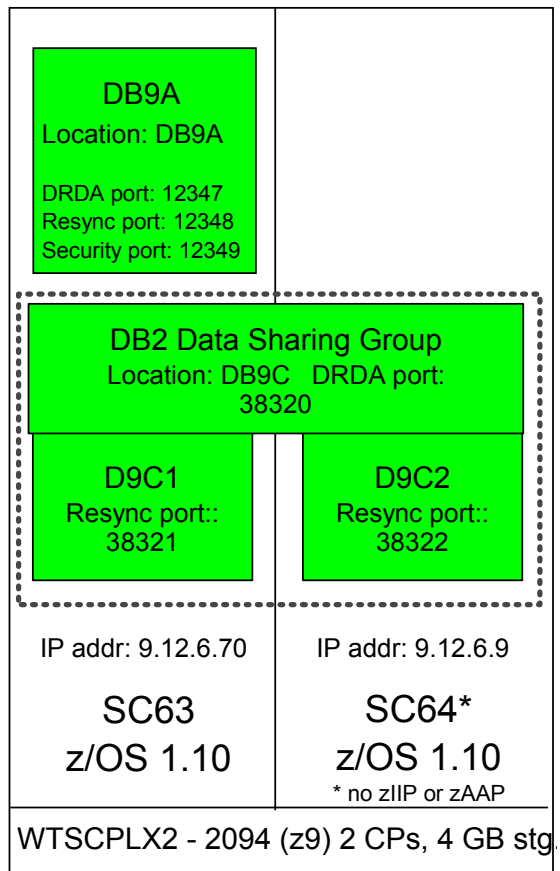


Figure A-1 DB2 configurations

## Stored procedures workload

We use a stored procedures workload. This provides a simple way of creating and driving a substantial workload on a DB2 for z/OS database.

The workload is composed of five principal tables (GLWTDPT, GLWTEMP, GLWTPRJ, GLWTPJA and GLWTEPA), 12 supporting tables, 16 Stored Procedures (DPTADD, DPTUPD, DPTUPR, DPTMGR, DPTLCK, DPTBAL, DPTDEL, EMPADD, EMPUPD, EMPUPR, EMPDEL, EMPQRY, EMPQR2, EMPFND, PRJADD, and PRJUPD), and 12 supporting procedures. It exploits views, referential integrity, partitioned tables, triggers, and so on. Refer to Figure A-2 to view the data schema.

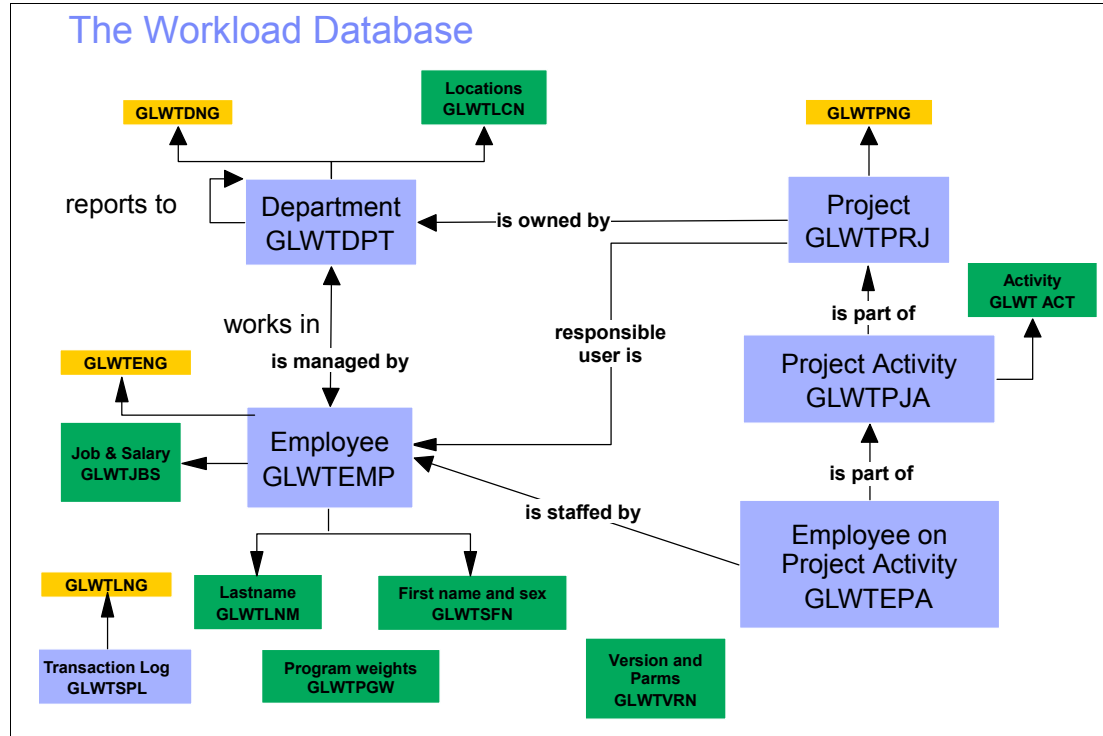


Figure A-2 The GLW database

Table A-1 summarizes the characteristics of the tables in our GLW database.

Table A-1 GLW table profiles

Table name	View name	Content	Number of columns	Number of rows (before workload)	Number of rows (after 10 minutes of workload)
GLWTDPT	WLDEPT	Departments	10	0	588
GLWTEMP	WLEMP	Employees	19	0	7203
GLWTPRJ	WLPROJ	Projects	13	0	1171
GLWTPJA	WLPROJACT	Activities on each project	10	0	21078
GLWTEPA	WLEMPPROJACT	Employees assigned to each activity on each project	10	0	68184
GLWLCN	WLLOCN	Departmental locations	2	130	130
GLWJBS	WLJOBSALARY	Employee jobs and salaries	4	15	15

Table name	View name	Content	Number of columns	Number of rows (before workload)	Number of rows (after 10 minutes of workload)
GLWTLNM	WLLASTNAME	Employee last names	2	351	351
GLWTSFN	WLSEXFIRSTNAME	Employee sex and first names	3	84	84
GLWACT	WLACT	Activities	3	18	18
GLWTDNG	WLDEPT_NO_GEN	Number generator for GLWTDPT	2	0	0
GLWTENG	WLEMP_NO_GEN	Number generator for GLWTEMP	2	0	0
GLWTPNG	WLPROJ_NO_GEN	Number generator for GLWTPRJ	2	0	0
GLWPNG	WLTRAN_NO_GEN	Number generator for GLWTSPL	2	0	0
GLWTSPL	WLSPLLOG	Log of each stored procedure CALL by the driver	6	0	14696
GLWTPGW	WLPROGWT	Runtime weighting for each stored procedure	3	119	119
GLWTVRN	WLVERSN	Database version control table	24	1	1

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

## IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 446. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Considerations for Multisite Sysplex Data Sharing*, SG24-7263
- ▶ *DB2 UDB for z/OS: Design Guidelines for High Performance and Availability*, SG24-7134
- ▶ *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465
- ▶ *DB2 for z/OS: Data Sharing in a Nutshell*, SG24-7322
- ▶ *DB2 for z/OS and OS/390 : Squeezing the Most Out of Dynamic SQL*, SG24-6418
- ▶ *A Deep Blue View of DB2 Performance: IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SG24-7224
- ▶ *IBM DB2 Performance Expert for z/OS Version 2*, SG24-6867
- ▶ *Implementing IBM Lotus Domino 7 for i5/OS*, SG24-7311
- ▶ *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-7270

## Other publications

These publications are also relevant as further information sources:

- ▶ *DB2 Version 9.1 for z/OS Administration Guide*, SC18-9840
- ▶ *DB2 Version 9.1 for z/OS Application Programming Guide and Reference for Java*, SC18-9842
- ▶ *DB2 Version 9.1 for z/OS Application Programming and SQL Guide*, SC18-9841
- ▶ *DB2 Version 9.1 for z/OS Codes*, GC18-9843
- ▶ *DB2 Version 9.1 for z/OS Command Reference*, SC18-9844
- ▶ *DB2 Version 9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845
- ▶ *DB2 Version 9.1 for z/OS Installation Guide*, GC18-9846
- ▶ *DB2 Version 9.1 for z/OS Internationalization Guide*, SC19-1161
- ▶ *DB2 Version 9.1 for z/OS Introduction to DB2 for z/OS*, SC18-9847
- ▶ *DB2 Version 9.1 for z/OS Messages*, GC18-9849
- ▶ *DB2 Version 9.1 for z/OS ODBC Guide and Reference*, SC18-9850
- ▶ *DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide*, SC18-9851
- ▶ *DB2 Version 9.1 for z/OS RACF Access Control Module Guide*, SC18-9852
- ▶ *DB2 Version 9.1 for z/OS Reference for Remote DRDA Requesters and Servers*, SC18-9853

- ▶ *DB2 Version 9.1 for z/OS SQL Reference*, SC18-9854
- ▶ *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855
- ▶ *DB2 Version 9.1 for z/OS What's New?*, GC18-9856
- ▶ *DB2 Version 9.1 for z/OS XML Guide*, SC18-9858
- ▶ *IBM OmniFind Text Search Server for DB2 for z/OS Installation, Administration, and Reference Version 1 Release 1*, GC19-1146
- ▶ *IBM Spatial Support for DB2 for z/OS User's Guide and Reference Version 1 Release 2*, GC19-1145
- ▶ *IRLM Messages and Codes for IMS and DB2 for z/OS*, GC19-2666
- ▶ *z/OS V1R10.0 MVS Setting Up a Sysplex*, SA22-7625

## Online resources

These Web sites are also relevant as further information sources:

- ▶ *Asynchronous DB2 data sharing locks not counted*, found at:  
<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/FLASH10632>
- ▶ InfoSphere Replication Server products  
<http://www.ibm.com/developerworks/data/roadmaps/qrepl-roadmap.html>
- ▶ Trade6 workload and description  
<https://www.software.ibm.com/webapp/iwm/web/preLogin.do?source=trade6>

## How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)



# Index

## Numerics

00C90088 151, 224, 282  
00C9008E 113, 224  
-911 152, 155  
-913 152, 155

## A

access 4, 14, 16, 44, 51, 67, 69, 87–88, 162, 211, 220, 222, 226, 267, 308, 369, 384  
access path 29, 85, 88–89, 91, 184, 189, 211, 262  
accounting 57, 61, 214, 221–222, 227–228, 265, 270, 284, 344, 372  
accounting record 240, 251, 264, 293, 404  
accounting report 60, 221, 237–238, 240, 284–285, 374, 383  
accounting trace 227, 265  
    class 3 228  
ACCUMACC 236  
ACQUIRE 16, 54, 66, 72, 100, 137, 180, 199, 273, 297–298, 328  
ACQUIRE(ALLOCATE) 100  
ACQUIRE(USE) 100  
affinity 430  
    isolating a set of tables 430  
    isolating at partition level 430  
ALL 55, 119, 163, 177, 230, 252, 263, 271, 273, 278, 390  
ALTER TABLE 4, 114, 191, 195, 301  
ALTER TABLESPACE statement 137, 191  
analysis 215, 231–232, 253, 256, 372, 403  
AND NOT 79  
application xxi, 4, 14, 41, 44, 63, 65, 67, 87–88, 162, 204, 206, 208, 221, 256, 260, 263, 267, 332, 339, 382, 417  
application code 69  
application design 87, 102, 207  
application logic 118, 121  
application program 10, 48, 52, 74, 78, 125, 128, 199, 204  
architecture 312  
attachment 123, 146, 152, 191, 208  
attribute 43, 46–47, 81, 102–103, 191, 240, 336  
authorization 51–53, 84, 146, 164, 184, 225, 231, 249–250, 256–257  
auxiliary table 137, 195–197  
AVG 236, 285–286, 383, 390

## B

BACKUP 421  
base table 25–26, 73, 76, 118, 127, 177, 187, 194  
    row 131, 188–189  
batch application 121, 126, 135, 150, 167, 208  
batch environment 391  
batch job 51, 120, 363  
batch program 122, 430

BIND 4, 9, 12, 19–21, 52, 72, 102, 122, 137, 180–181, 206, 232, 236, 262, 285, 326, 383, 392, 426, 432  
bind 23, 46, 51–52, 68, 88, 90, 180, 262, 326, 432  
BIND parameter  
    ACQUIRE 100  
    RELEASE 101  
BIND PLAN 181, 184  
block fetch enablement 38  
BPOOL 286, 407  
buffer pool xxi, 56, 60, 67, 75, 159–160, 195, 239, 311, 314, 316, 366–367, 371, 430–432  
buffer pools 371

## C

C 45, 59, 93, 99, 243, 246, 257, 318, 366  
CACHE 79–80, 181, 429  
CACHE size 430  
caching 4, 23, 107, 137, 145, 180, 316, 340  
CAF 11, 48, 123, 144, 151, 183, 206, 208  
CASCADE 405  
catalog and directory 51–52, 128, 164  
catalog table 35, 53, 70, 164, 192, 224, 396  
CF 221, 237, 265, 285, 309–310, 371, 426, 431  
CHECK DATA 165, 178, 194  
CICS 4, 11, 23, 48, 123–124, 183–184, 206, 404  
CICS attachment facility 152  
claim class 54, 163, 208  
claims xxi, 53, 91, 162, 208, 242, 256–257, 339  
claims and drains 54–55, 176, 233, 269  
class 49, 75, 162, 208, 214, 220, 265, 279, 313, 372, 383  
CLUSTER 84, 86, 240, 399, 420  
clustering 81  
clustering index 74, 81–82, 84, 240  
clustering process 154  
code table 157  
COLLID 243–244, 283  
COMMIT 12, 20, 22, 46, 48, 70, 90–91, 101, 120–121, 123, 155, 169, 180, 210, 221, 228–229, 264–265, 271, 326, 373, 426, 431  
commit frequency 121, 125, 152, 206, 241, 247  
    recommendation 125  
Commit Log Sequence Number 36  
compatibility rules 55, 164–165  
components 100, 309, 369, 441  
compression 38, 66, 75  
concurrency xxi, 4, 7, 14, 17, 52, 63, 65, 84, 87, 119, 161, 166, 201, 210, 217, 219–220, 255–256, 306, 315, 394  
concurrency problems 52, 67, 72, 220–222, 255–256, 265, 280  
concurrent updates 97, 106, 114  
condition 31, 103, 117, 172, 213, 224, 253, 261, 348  
CONNECT 243–244, 271–273, 277, 343, 404–405  
connection 145, 180, 207–208, 223, 229, 248–249, 256, 353, 371

considerations 19, 50, 65–66, 72, 87, 96, 107, 179, 201, 339, 392, 418  
 console messages 221–222, 226, 238, 284  
 constraints 36, 53, 72, 94–95, 130, 134, 191, 209–210  
 contention 52, 57, 59, 69, 73, 106, 108, 112, 175, 206, 209, 219, 230–231, 265, 269, 279, 310, 365, 376  
 CONTOKEN 243–244, 283  
 control 7, 10, 20, 23, 51–52, 75, 87–88, 100, 161–162, 202–203, 306, 365, 394  
 control table 77–78  
 COPY 4, 12, 55–56, 86, 169–170, 194, 431  
 COUNT 30, 119, 248–249, 289, 346, 368  
 COUNTER 243, 246–247, 282  
 Coupling Facility 310–311, 372, 389  
 coupling facility 195, 307–308, 365, 432  
 CPU time 49, 70, 210, 227–228, 230, 233, 277, 284, 341  
 CPU usage 85, 156, 221, 233  
 CREATE TABLE 72, 114, 127–128, 191  
 CREATE TABLESPACE statement 67, 75, 420  
 CREATOR 244, 246, 278  
 cross invalidation 429  
 CS 8, 21, 27, 46, 54, 69, 77, 88–89, 162, 166, 210–211, 232, 257, 259, 273, 392  
 CS isolation 106, 112, 188, 426  
 CURRENT SQLID 235  
 CURRENTDATA 8, 27, 29, 34, 69, 90, 112, 184, 186, 232, 392, 426  
 cursor stability 21, 101, 103, 127, 144, 162, 186, 212, 257  
 CURSOR WITH HOLD 57, 91, 144  
 CYCLE 207, 390

## D

data access 29, 85, 89, 113, 123, 211, 334  
 data clustering sequence 82  
 data consistency 82, 90, 135, 306, 315  
 data inconsistency 109  
 data integrity xxi, 1, 4, 14, 25, 56, 73, 77, 87, 103, 187, 193, 306  
 data page 29, 57, 71, 74, 81, 107, 110–111, 115, 247, 283, 326, 329, 394  
 data row 57, 72, 90  
 data set 68, 163, 231, 253, 276, 315, 385  
 data sharing 4, 14, 26, 44, 58, 71, 75, 78–80, 88, 95, 102, 202–203, 228, 234, 239, 267, 269, 305, 365, 417, 430–431  
 data sharing group 44, 148–149, 208, 240, 269, 305–306, 366, 374, 431  
 data type 301  
 database design 63, 65, 150, 158  
 Database name 283  
 DATE 243, 246–247, 251, 271, 273, 278, 282, 390, 412  
 DB2  
     controls 87, 236, 326  
 DB2 attachments 123  
 DB2 commands 41, 53–55, 146, 161, 179, 193, 208, 255, 346, 366  
 DB2 member 242, 306, 308, 380, 384, 426, 429  
 DB2 subsystem 26, 51, 147, 152, 190, 204, 225, 236, 241, 253, 306

DB2 table 28, 63, 81  
 DB2 trace records 276, 284  
 DBAT 48, 260  
 DBD 15, 22–23, 44, 52–53, 72, 80, 108, 113, 190, 257–258, 263, 315, 318, 366–367  
 DBD01 52, 193  
 DBNAME 244, 246, 278, 368  
 DBRM 100, 190, 199, 228  
 DDF 27, 50, 160, 212, 229, 233, 236  
 DDF exception conditions 227  
 DDL 12, 23, 63, 80, 86, 161, 191, 194, 285, 301, 385, 398, 430  
 deadlock cycle 49  
 deadlock detection cycle 204, 207, 231, 360  
 deadlock situation 153, 245, 361  
 deadlock time 48, 207, 288, 295–296  
 deadlocks 4, 14, 17, 26, 32, 48–49, 57, 69, 76, 78, 91, 121, 126, 144–146, 172, 177, 182, 202, 205, 207, 220, 222–223, 227, 231, 255, 261, 268, 280, 282, 284, 310, 314–315  
 decimal 224, 244  
 DECLARE 22, 27, 74, 251, 298  
 DECLARE CURSOR 27  
 declared temporary table 26, 74, 157  
 default value 70, 205, 207–209, 225, 227, 248  
 DEGREE 189, 235, 242, 285, 392  
 DELETE 4–5, 14, 19, 46, 88–89, 165, 168, 210, 285–286, 289, 333, 382, 392  
 delete 18, 66, 72, 75, 88, 181, 187, 210–211, 333, 353  
 dependent table 95, 154, 394, 420  
 DESCRIBE 149, 285–286, 393, 395  
 design 14, 19, 63, 65, 87, 207, 209, 322, 394, 403, 417  
 details 427  
 DISPLAY DATABASE LOCKS 259, 318, 320, 366  
 DISPLAY DATABASE LOCKS command 320, 322  
 DISPLAY THREAD 259–260  
 DISTINCT 35, 127–128  
 distributed environment 12, 124, 144, 186  
 domain 9  
 DPSI 54, 76  
 DRAIN ALL 177  
 drains xxi, 53–54, 108, 122, 157, 159, 161–163, 233, 238, 242, 257, 259, 269, 369  
 DRDA 27, 33, 79, 144, 180, 189, 229, 271–273  
 DROP TABLE 263  
 DSC 262, 264  
 DSNIO31I 225  
 DSNJO31I 427  
 DSNR035I 226, 249, 427  
 DSNT397I 113, 257–258, 318, 366–367  
 DSNT408I 113, 119, 263  
 DSNT500I 129, 197, 233, 256, 263  
 DSNT501I 197, 223–224, 282  
 DSNZPARM 15, 71, 90, 101–102, 125, 184, 202, 204, 225–227, 241, 247, 340, 372, 374, 432  
     IMMEDWRI 432  
     IRLMRWT 125, 204–205  
     UTIMOUT 125, 206  
 DSNZPARMs 147, 201, 204, 214, 228, 248, 336  
 DTT 157

dynamic prefetch 61, 154  
dynamic scrollable cursors 141  
dynamic SQL 15, 20, 53, 108, 137, 144, 180, 251, 263  
dynamic statement cache 24, 108, 251–252, 256, 262  
DYNAMICRULES 273

## E

environment 11–12, 20–21, 53, 71, 87, 95, 169, 202, 206, 219, 226, 228, 267, 269, 305, 365–366  
EXCLUDE 270, 276, 280  
exclusive 4, 14–16, 52–53, 88, 95, 106, 182, 232, 235, 279, 310, 313, 368  
EXEC SQL 38  
EXPLAIN 138, 141, 252, 262, 392  
Explicit hierarchical locking 325  
explicit hierarchical locking 325  
expression 35  
external 284, 390  
EXTRACT 237, 285, 405  
extract 262–263, 404

## F

false 314, 343, 376–377  
FETCH 34, 38, 47–48, 78–79, 91, 93, 95, 153, 167, 186, 285–286, 369, 393  
fetch 38, 46, 61, 90, 95, 186, 210, 227, 232  
FOR FETCH ONLY 148, 152, 188  
FOR READ ONLY 34–35  
FOR UPDATE 27, 29, 35, 78, 88, 95, 98–99, 186, 210  
FOR UPDATE OF 34, 36, 89, 97, 101, 153, 186  
forced write 121  
foreign key 53, 191, 301–302  
FREE LOCATOR 285  
free space 66, 71, 174, 332–333, 389  
FREEPAGE 39, 66, 71  
function 4, 12, 49, 51, 120, 191, 233, 329

## G

GBP-dependency 431  
GENERATED ALWAYS 115–116  
GET DIAGNOSTICS 150–151, 155  
global lock contention 353  
global locks 306, 314  
GRANT 12, 52, 70, 84, 157, 191–192, 235, 242, 285, 300  
GROUP BY 35, 266  
Group restart 353  
group restart 353

## H

H 99, 251, 258, 318, 366  
handle 4, 51, 77, 106, 120, 190, 296, 332, 391  
health check 228  
hole 57  
hot data page 81  
hot pages 65, 77, 81, 152  
hot rows 81–82  
hot spots 80–81, 139, 401  
hybrid state 17

## I

I xxi, 7, 12, 38, 45, 76, 85, 122–123, 183, 206, 208, 221, 264, 318, 366  
IBM System z 4, 284  
identifying 71  
identity column 79, 155, 194, 429  
IFCID  
    172 223, 243, 282, 288  
    3 225–226, 250, 279, 282, 404  
    313 226, 247–248  
IFCID 172 220, 227, 243, 286, 288  
IFCID 196 220, 223, 247  
IFCID 226 239, 279  
IFCID 227 239  
IFCID 313 226, 247–248  
IFCID 337 221, 225, 250  
IFCID 56 235, 279  
IFCID 57 235  
IFI 146, 237, 248, 285, 383, 405  
illustration 6–7, 32–33  
image copy 169, 195, 332  
IMMEDWRI 432  
IMMEDWRITE 273, 432  
IMS 4, 11, 23, 48, 123–125, 152, 183, 203, 206, 272  
index 15, 26, 42–43, 69, 72–73, 75, 89–90, 162, 178, 193–194, 197, 210–211, 232–233, 239, 256, 315, 333, 367–368, 389, 410  
index access 81, 85, 104, 185  
index key 60, 76, 140  
index look-aside 122–123  
index scan 29, 46, 86, 91, 329  
INSERT 4–5, 14, 24, 26, 46–47, 71, 82, 84, 86, 88–89, 120, 174, 181, 252, 273, 285–286, 297, 333, 381–382, 431  
insert 25, 42–43, 65, 71, 92, 174, 212, 262, 333, 399, 402  
INSERT INTO SELECT 429  
INSERT statement 92, 416  
installation 48, 70, 95, 137, 172, 202, 204, 224, 252, 334  
INSTANCE 243, 246–248, 270–271, 412–413  
intent exclusive 16–17  
interaction 125, 133, 161, 191, 202, 309, 349  
interaction integrity 125  
inter-DB2 R/W interest 431  
intermediate commit points 125, 206  
Internal Resource Lock Manager 44  
IPROC 122  
IRLM 12, 14, 29, 41, 44, 69–71, 90, 96, 119, 122, 167, 169, 202, 221, 230, 264–265, 267, 306, 370, 372, 426  
IRLM resource wait timeout 207  
IRLMRWT 125–126, 168, 204, 206  
IRLMRWT DSNZPARM  
    value 125  
IS 15–17, 45, 53, 68, 89, 167, 181, 205, 207, 222–223, 225, 246, 261, 282, 310–311, 317, 366  
ISOLATION 9, 21, 42, 46, 69, 77, 90–91, 101, 175, 184, 210, 232, 273–274, 289, 392, 426, 428, 431  
ISOLATION CS 94  
isolation level 7–8, 19, 28, 35, 77, 88, 93, 101, 184, 211–212, 232  
isolation repeatable read 431

ISOLATION RR 95  
ISOLATION UR 108, 127  
IX 15–17, 45, 53, 68, 89, 167, 170, 258, 271, 310–311,  
366–367

## J

Java 4

## K

KEEPDYNAMIC 24–25, 105, 107, 181, 273, 392  
keyword 35, 224, 253, 266, 288, 342, 366–367

## L

latch contention 57, 59, 73, 86, 230–231, 234, 265, 279,  
362, 385  
latch wait time 238–239, 284, 286  
latches 4, 56, 77, 108, 139, 230, 234–235, 279, 314, 382,  
384  
LENGTH 113, 257–258, 318, 366–367  
let 45, 113, 117, 169, 266, 290, 404, 411  
level 2 326, 344  
LIKE 264  
LIST 287, 343, 368  
list 6, 21, 50, 53, 76, 129, 159, 181, 195, 211, 214, 260,  
278, 295, 311, 370  
L-locks 240, 257, 310, 366, 375, 428  
LOAD 56, 71, 126, 164, 174, 194–195  
LOAD utility 72, 173, 175  
LOB 14, 16, 67, 90, 95, 162, 180, 194, 221, 237, 257,  
265, 274, 383, 405  
LOB locks 25, 68, 135  
lock avoidance 18, 28, 44, 47, 66, 77, 90, 112, 122, 167,  
186, 232, 276, 363, 392, 419, 426  
lock contention 73, 81–82, 155–156, 175, 209, 238, 240,  
310, 380–381  
lock duration 75, 82, 87, 91, 96, 100, 122, 152, 186, 258  
    recommendation 102  
lock escalation 15, 24, 30–31, 48, 69–70, 90, 92, 105,  
122, 176, 181, 209, 220–221, 225, 227, 250–251, 262,  
348  
lock granularity 14, 101  
lock management 4, 14, 49, 71, 310, 322  
lock mode 16, 18, 29, 94, 97, 167, 186, 262, 310–311  
lock modes 58  
lock negotiation 84, 234, 311, 316, 380  
lock size 14, 16, 68, 90–92, 332  
lock suspension 32, 48–49, 86, 134, 150, 230, 235, 247,  
267–268, 270, 385, 402  
LOCK TABLE 16–17, 27, 66, 69, 88, 105, 181, 235, 242,  
285–286, 328, 346, 407  
LOCK TABLE IN SHARE MODE 101, 106  
LOCK TABLE statement 24, 26, 105, 189  
locking xxi, 4, 14, 26, 41–42, 53, 56, 63, 65–66, 87–88,  
90, 101–102, 146, 154, 167, 182, 201–202, 217,  
219–220, 222, 227, 255, 265, 268, 305, 310–311, 365,  
388, 391, 426, 428  
locking control options 27  
locking implementation 115–116

locking problems 52, 65, 143, 150, 228  
locking scenario 114, 140, 395–396  
LOCKMAX 48, 66, 69, 90, 176, 209, 251–252, 348  
LOCKMAX option 69  
LOCKMAX SYSTEM 85, 252  
locks 4, 11, 14, 41, 63, 65–66, 81–82, 84, 87–88, 100,  
112, 153, 162–163, 202–204, 219–222, 256–258, 306,  
365–366, 426, 431  
LOCKSIZE 16, 24, 66–67, 88–89, 187, 209, 328–329,  
382, 394  
LOG 59, 61, 166, 195, 213–214, 221, 225, 264, 383  
log 23, 73, 121–122, 156, 166, 177, 195, 197, 206,  
213–214, 225, 247–248, 284, 331, 340, 415, 429  
log buffer  
    forced write 122  
log records 32, 61, 195, 213–214, 225, 248–249, 334  
logging 12, 23, 49, 74, 126, 130, 213–214, 225, 253, 279,  
415  
logical locks 14, 314  
logical unit 5, 65, 120, 123, 148  
    previous updates 5, 120  
logical unit of work 5, 11, 120, 148  
LPL 160, 195  
LRDRTHLD 212, 247–248, 250

## M

maintenance 73, 85, 177, 334, 442  
mass delete lock 112, 428  
materialize 38–39, 91, 131  
materialized query table 35  
maximum number 69–71, 90, 122, 204, 241, 354, 380  
maximum number of locks 70–71, 122, 204, 210  
maximum number of page or row locks 209  
MAXROW 1 429  
MAXROWS 1 85, 394, 419  
MEMBER CLUSTER 421  
memory 57–58, 79, 145, 181, 204, 314  
MERGE 4–5, 14, 24, 46–47, 118, 285, 392, 395  
MODIFY 171, 195, 203, 274, 342, 412  
MONITOR 261–262  
MQ 28  
MQT 73, 127  
multiple batch job steps 120  
multiple DBMSs 12  
multiple rows 155, 406

## N

next value 78  
NO CACHE 79–80, 429  
Non-GBP-dependent 431  
NOT IN 79  
NULL 115, 290, 301  
null value 137  
NUMLKTS 134–135, 176, 208  
NUMLKUS 48, 71, 147, 209, 241  
NUMLKUS threshold 122

## O

Object 25, 53, 194, 283  
online xxiii, 4, 12, 53, 63, 85, 113, 117, 120, 166,  
205–206, 222, 227, 252–253, 280, 366, 376  
ONLINE LOAD 429  
OPEN CURSOR 91, 122, 140, 188  
open cursor 91, 122, 159, 188  
OPEN/CLOSE 221, 237, 264, 285, 383, 405  
optimistic locking 114–115, 117, 222  
options 16, 19, 48, 65, 116, 119, 176, 202, 232, 252, 256,  
296, 341, 371  
OR 113, 119, 130, 264  
ORDER 35, 77–80, 244, 266, 284, 383, 429  
ORDER BY 153  
ORDER table 82  
ORDER\_SUMMARY table 82–83  
outer subselect 35

## P

package 27, 35, 51–52, 88, 90, 99, 181, 221, 228, 239,  
244, 262, 265, 272, 385, 404, 426  
package level accounting 286, 407  
page level 29, 45, 56, 65, 69, 114, 117, 125, 154, 170,  
262, 291, 331–332, 394, 396  
page lock 14, 18, 36, 67, 69, 71, 138, 160, 185  
page physical locks 315  
page P-locks 314–315, 329, 376, 380, 386, 420  
page set 54, 162, 182, 231, 257, 311, 366, 430–431  
page set P-lock negotiation 316  
page set P-locks 319, 322, 386  
page size 67, 75  
Parallel Sysplex  
    environment 417  
parallelism 38, 58, 61, 72, 85, 188–189, 203, 231  
parent table 72, 95, 195  
PART 16, 66, 93, 105, 166, 240, 257–258, 318, 366–367  
PARTITION 240, 291, 297–298, 368, 421, 431  
partition independence 76, 85, 173, 175, 367  
PARTITIONED 443  
partitioned 15–16, 25, 54, 66, 162, 251, 315, 366  
partitioned table space 31, 54, 56, 66–67, 98, 172, 251,  
283, 328  
partitioning 54, 175, 195–196, 430  
partitions 16, 20, 44, 54, 66, 98, 162, 256, 328, 367  
PATH 285, 426  
PCTFREE 39, 66, 71  
PERFORMANCE 243, 246–247, 266, 271, 373  
performance xxi, 6, 14, 25, 60, 63, 65, 67, 87, 96, 175,  
184, 194, 204, 210, 213–214, 220–221, 225, 227,  
279–280, 306, 322, 371  
Performance Expert 37, 228, 373  
physical design 82  
physical locks 14, 257, 315, 376  
PK62027 25  
PK62214 424  
PK65220 422  
PK77514 58  
PK81471 422  
PK85159 376, 387

PK85543 376, 387  
PK93904 248–249  
PK96263 93  
plan 19–20, 24, 51, 66, 72, 88, 99–100, 181, 221, 223,  
227, 239, 242, 262, 265, 328, 331, 404, 426  
PLAN\_TABLE 262  
P-locks 60, 75, 85, 102, 122, 138–139, 234, 239–240,  
257, 259, 269, 310–311, 315, 366–367  
populate 157  
positioned UPDATE 35, 117, 131  
PREFETCH 82, 286–287, 406–407  
prefetch 61, 154, 260  
PREPARE 9, 24, 28, 35, 118, 149, 181, 251, 285–286,  
393, 395  
prevent locking problems 155  
preventing 5, 54–55, 157, 163  
primary key 36, 53, 83, 191  
PRIQTY 86  
privilege 24, 181, 192  
problems xxii, 4, 43, 51–52, 65, 67, 72, 87, 117, 139, 190,  
217, 219–220, 255, 347, 392  
program logic 82, 147  
propagation 84, 234, 309, 368, 376, 426  
PT 128, 144, 259

## Q

QMF 206  
QUERY 110, 149, 230, 241, 273–274, 377, 388  
query 8, 21, 43, 46, 73, 85, 91, 101, 187, 203, 211, 231,  
244, 278  
QUIESCE 59, 170, 286, 346, 407

## R

R/O 316, 431  
R/O switch 431  
R/W 316  
R/W interest 431  
RANDOM 76, 399  
RBDP 178, 196  
read and write 20, 22, 56, 76, 85, 197, 210  
read stability 21, 28, 101, 103, 159, 162, 210, 212  
read-only 17–18, 56, 65, 85, 88, 90, 171, 186, 210, 213,  
316  
read-only cursor 34, 93, 130, 140  
real 55, 60, 235, 247, 279–280, 343, 353, 374  
REASON 113, 119, 224, 263, 390  
reason code 00C90088 224  
reason code 00C9008E 224  
reason codes 224, 272, 413  
reasons for locking 34  
REBIND 19–21, 52, 141, 180–181  
REBUILD INDEX 175, 178, 194–195  
REBUILD-pending 196  
recommendations 84, 152, 155, 341  
RECOVER 55, 163, 194, 429  
recovery xxi, 5, 9, 11, 26–27, 85, 120–121, 175, 194,  
199, 212–213, 225–226, 231, 247, 313, 334  
Redbooks Web site 446  
    Contact us xxiii

- reducing network traffic 144, 189
- referential integrity 5, 34, 36, 72, 94, 130
- RELEASE 20, 46, 70, 72, 88, 90, 170, 180, 199, 221, 231, 236–237, 264, 272–273, 285, 326, 383
- RELEASE(COMMIT) 101
- RELEASE(DEALLOCATE) 101
- REORG 4, 12, 38–39, 55, 71, 113, 117, 163, 166, 194, 232, 394
- repeatable read 4, 6, 21, 42, 54, 86, 101, 103, 106, 162–163, 210, 315
- REPORT 174, 194, 236, 241, 253, 259–260, 373
- resource consumption 426
- resource lock manager 56, 70, 309
- resource manager 120
- RESOURCE TYPE 302 224
- RESOURCE TYPE D01 224
- resource unavailable 129, 178, 208, 224, 256, 261, 263, 348
- RESTART 122, 194, 352, 414
- restart 4, 122, 156, 197, 206, 208, 213–214, 311, 334
- RESTRICT 34, 256, 430
- restricted states 161, 193–194, 198–199, 256
- result set 48, 101–102
- result table 34–35, 140–141, 186
- retained locks 206, 349
- return 4, 11, 38, 48, 90, 211, 272, 356, 413
- RI 34, 195, 274, 276, 329
  - DB2 enforced 154
- RID 38, 116–117, 173, 211
- RMF reports 376, 389
- RO 24–25, 107, 179, 182, 197–199, 213, 316
- ROLLBACK 11, 48, 100, 109, 120–121, 123, 131, 147, 155, 183, 211, 229, 236, 268, 285, 383, 405, 432
- row level 15–16, 26, 56–57, 65, 81, 90, 122, 154, 168, 176, 239–240, 278, 314, 329, 381, 386, 420
- row lock 14, 17–18, 57, 66, 69, 89, 103, 419, 426
- row lock mode
  - X 134
- row locking 42, 56, 69, 329, 332, 380, 394
- ROWID 116
- RR 7, 21, 24–25, 42, 78, 86, 88–90, 101, 162, 167, 210, 250, 259, 329, 426, 431
- RR isolation 106, 142
- RRS 157
- RRSAF 27, 123, 157, 191, 206, 236, 259
- RS 8, 27, 46, 78, 88–90, 162, 184, 210
- RS isolation 106, 210, 426
- run time 142, 184, 187
- RUNSTATS 39, 55, 113, 178, 181

## S

- S 14, 16, 42–43, 45, 66, 68, 89–90, 166, 170, 210, 225, 243, 246, 258, 308, 366–367
- same data 4, 14, 44, 87, 117, 126, 329
- same table 21, 35, 42, 69, 85, 106, 130, 164, 166, 268, 327, 357, 389
- same time 5, 7, 14, 32, 79, 87, 117, 139, 163–164, 242, 267, 273, 292, 329, 339, 384
- SAVEPOINT statement 121
- savepoints 120–121

- SCHEMA 286, 407
- schema 39, 84, 283, 300–301
- scrollable cursor 91, 117
  - locking 117
- SECQTY 86
- segmented 15–17, 66, 68, 90, 162, 178, 244, 319
- segmented table space 24, 67, 84, 90, 93–94, 181, 278, 327
- SELECT 4–5, 12, 14, 24, 42–43, 47, 77, 88–89, 101, 165, 167, 210, 244, 246, 264, 278, 366, 369
- SELECT FROM INSERT 132
- SELECT statement 30, 35, 79, 119, 132, 252, 294
- SELECT SUBSTR 246
- SENSITIVE STATIC 35
- SEQ 286, 301, 404–405
- sequence 9, 26, 45, 58, 69, 74, 78–79, 100, 120, 183, 251, 270, 280, 287, 299, 324, 355
- sequence number 78, 155, 363, 429
  - first request 429
  - next request 429
- sequential detection 122
- sequential number 78, 84
- serialization mechanisms xxi, 12
- SET 10, 30, 117, 132, 167, 203, 235, 242, 277, 285, 342, 368
- SHARE 17, 38, 101, 103, 105, 186, 210, 429
- side 140, 244, 246, 267–268, 272, 324, 377, 390
- simple 12, 32–33, 60, 66–67, 106, 130, 142, 280, 368–369
- single row 14, 66, 85, 96–97, 402
- single table 69, 141, 403
- SKCT 51–52, 92, 128–129, 191, 380, 387
- SKPT 51–52, 144, 191, 271, 380, 387
- S-lock 22–23, 43, 77, 107, 124, 186, 210–211, 283, 291, 293, 353
- SMF 214, 235, 239, 265, 269, 385–386
- SMFACCT 214, 228
- SMFSTAT 215, 228
- sort 13–14, 234, 293, 371, 412
- space map 57, 84, 240, 314, 380, 386
- space map pages 240, 329, 381, 401
- SPUFI 112, 225, 251–252, 259, 264
- SQL 4–5, 15, 42, 47, 68, 70–71, 87–88, 100, 120, 123, 161, 209–210, 225, 229, 235, 262, 366, 368, 428
- SQL design 133
- SQL statement 27, 35, 37, 42, 48, 78, 88, 167, 247, 251, 262–263, 366, 395
  - execution 148
  - locking 98, 168, 262, 270
- SQL statement text 252, 262, 290
- SQLCA 150–151, 274–275, 293
- SQLCODE 48, 90–91, 98, 155, 172, 188, 208, 263, 274–275
- SQLSTATE 113, 263, 274
- START DATABASE 12, 193, 195, 197
- STATEMENT 225, 251, 263, 273–274, 414
- statement 12, 15–16, 19, 42, 66–68, 75, 88, 167, 211, 225, 247, 250, 256, 262, 342, 366, 420
- STATIC 35
- static scrollable cursor 117

static SQL 25, 107, 145, 180, 251, 290  
 STATIME 215, 227, 265, 372  
 statistics 58–60, 75, 85, 122, 178, 215, 220–221, 225–226, 265, 279, 282, 341, 344, 372  
 statistics record 234, 241–242  
 statistics trace 227, 265  
 STATUS 93, 113, 203, 205, 257–258, 318, 366  
 STMT 286, 407  
 stored procedures 50, 58, 144, 281, 283–284, 286, 383, 392  
 subquery 35  
 subselect 35  
 SUBSTR 244, 246, 264  
 subsystem object locking 51  
 SUM 266, 268, 408  
 summary 26, 82, 276, 278, 322, 387, 389  
 suspensions 4, 14, 23, 48, 60, 100, 149, 207, 221–222, 266–267, 354, 377  
 SYSCOPY 129, 164  
 SYSIBM 39, 52, 70, 79, 127, 224, 246, 278, 290, 294, 396, 429  
 SYSIBM.SYSDUMMY1 298  
 SYSIBM.SYSSEQUENCES 79, 429  
 SYSIBM.SYSTABLES 244, 246, 278  
 SYSLGRNG 221, 237, 264, 266, 285, 383, 405  
 SYSOPR 260–261, 415  
 sysplex 4, 231, 389

## T

table design 72  
 table function 35  
 table lock 16, 19, 66, 69, 74, 90, 112, 133, 246, 272, 345  
 table space 15, 66, 88, 163, 204, 206, 224, 232, 257, 307, 366, 368  
 table space lock 16, 25, 30, 68, 70, 90, 98–99, 242  
 table space scan 24–25, 69, 81, 91, 95, 210–211  
 tables 20, 44, 52, 65, 90, 164, 211, 213, 257, 301, 380–381  
 TCP/IP 50, 221, 237, 265, 285–286, 383, 405  
 temporary table 26, 74, 107, 140  
 thread 14, 17, 20, 46–47, 100–101, 122, 143–144, 206, 212, 223, 225, 227–228, 256, 259, 326, 333, 384  
 TIME 207, 221, 233, 237, 264, 273, 340, 371, 383  
 time out 54, 69, 72, 150–151, 204, 247, 268, 354  
 timeouts 18, 48, 57, 72, 94, 106, 112, 126, 168, 177, 204–206, 220, 223, 231, 233, 253, 261, 268, 280, 310, 314–315, 350  
 TIMESTAMP 92, 113, 243, 246–247, 271, 273, 412  
 traces 227–228, 255, 265, 344, 372  
 TRACKMOD NO 240, 399–400  
 triggers 35, 48, 65, 76, 95, 130, 174, 337–338, 360  
 TS 93, 197, 240, 257–258, 318, 366  
 TSO 11–12, 48, 93, 113, 183, 206, 225–226, 248–249, 257–258, 318, 366  
 TSO batch 12  
 two-phase commit 12, 23, 157  
 TYPE 93, 113, 203, 224, 233, 243, 257–258, 318, 366–367  
 type of table space 332

## U

U 16–17, 66, 78, 89–90, 184, 186, 210, 232, 243, 246–247, 262, 266, 310–311, 390, 408  
 UK38906 25  
 UK39357 424  
 UK41212 422  
 UK43998 58  
 UK49345 248–249  
 uncommitted read 17–19, 89, 103, 108, 111, 162, 168, 428  
 UNION 35  
 UNIQUE 78, 429  
 unique index 60, 77  
 unit of recovery 5, 9–11, 120–122, 156, 199, 213, 226, 248–249  
 unit of work 5, 21, 23, 26, 65, 74, 82, 84, 87, 91, 97, 101, 121, 123, 155, 162, 226, 276–277  
 UNLOAD 28, 166, 176  
 unordered updates 154  
 UPDATE 4–5, 14, 24, 46–47, 77–78, 82, 88, 120, 125, 167, 195, 210, 221, 236–237, 264, 274–275, 333, 366, 368, 429, 431  
 UPDATE statement 30, 118, 133, 167, 169  
 UPROC 122  
 UR 8, 26–27, 47, 69, 88–89, 97, 101, 112, 162, 165, 212, 225–226, 290, 294, 298, 301, 428  
 URCHKTH 212, 226, 247–249  
 URLGWTH 213, 225, 247–248  
 USER 127–128, 170, 204, 209, 225–226, 248–249, 290, 298, 420, 432

## V

VALIDATE 68, 392  
 VALUE 205, 207, 273–274, 407, 413  
 VALUES 88, 92, 290, 298  
 VARCHAR 38, 60  
 VARIABLE 235  
 variable 38, 157, 214, 232  
 VERSION 243, 246–247, 251, 266, 271, 273, 346, 373  
 victim 32  
 VIEW 192, 285  
 view 35, 76, 124, 203, 253, 276, 361, 391  
 VOLATILE 81

## W

wait time 27, 112, 204, 238, 264, 284, 286, 384  
 WebSphere 4, 12, 28  
 WHERE clause 19, 36, 108, 112, 125, 181, 211  
 WHERE CURRENT OF 188  
 WITH 27, 54, 86, 90–91, 165, 222–223, 261, 282, 346  
 WITH HOLD 22, 27, 57, 91, 122, 162, 183  
 WLM 49–50, 231, 404–405  
 work file 21, 71, 91, 108, 187  
 WRITE 54–55, 169, 213, 221, 237, 264–265, 285, 383, 402  
 write xxiii, 17, 20, 22, 56, 73, 76, 108, 121–122, 163–164, 210, 213, 222, 239, 257, 315, 366, 380  
 write claim class 54, 166

## **X**

X 14, 16–17, 45, 66, 68, 89–90, 168, 181, 210, 224, 226, 236, 243, 258, 262–263, 308, 366–367

XES 234, 273–274, 308, 372, 375

XES contention 353, 356, 377–378

X-lock 22, 26, 57, 77–78, 92, 210–211, 232, 245, 283, 291, 320

XML 14, 67, 95, 162, 194, 257

## **Z**

z/OS 4, 50–51, 56, 76, 131, 189, 202, 224, 261, 265, 306, 368





# DB2 9 for z/OS: Resource Serialization and Concurrency Control

(1.0" spine)

0.875" x 1.498"

460 <-> 788 pages







# DB2 9 for z/OS: Resource Serialization and Concurrency Control



**See how DB2 allows  
concurrent accesses  
to data with integrity**

**Understand the  
locking parameters  
for your applications**

**Tune interactions  
between applications  
and utilities**

Locking is the generic term used to refer to the database management system function that is required for managing interprocess concurrency and maintaining data integrity. However, locking is just one of the serialization mechanisms available in IBM DB2 for z/OS. DB2 uses different mechanisms for serialization to achieve its goal of maximizing concurrency without losing integrity with a minimum cost in CPU, I/O, and storage resources.

In this IBM Redbooks publication, we review and explore the different serialization mechanisms used in DB2, such as transaction (DML) locking, claims and drains, restrictive states, latching, and optimistic serialization.

This book was written for application developers in order to help them better understand serialization mechanisms and how they influence application design decisions.

## **INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION**

### **BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
**[ibm.com/redbooks](http://ibm.com/redbooks)**